UNIVERSIDAD POLITÉCNICA SALESIANA
SEDE QUITO

CARRERA: INGENIERÍA DE SISTEMAS

Tesis previa a la obtención del Título de: INGENIERO DE SISTEMAS

TEMA:

ANÁLISIS, DISEÑO Y DESARROLLO DE UN PROTOTIPO DE PROTOCOLO DE TRANSPORTE BASADO EN COMUNICACIÓN TCP CON CAPACIDAD DE CUBRIR LAS NECESIDADES DE TRANSFERENCIA DE DATOS SEGUROS, CONFIABLES Y DE ALTA DISPONIBILIDAD

AUTORES:

JORGE EDUARDO CHAPACA GARZÓN

JAIRO DAVID ROJAS BUSTAMANTE

DIRECTOR:

DANIEL GIOVANNY DÍAZ ORTÍZ

Quito, diciembre de 2013

DECLARATORIA DE RESPONSABILIDAD Y AUTORIZACIÓN DE USO DEL TRABAJO DE GRADO

Nosotros Jorge Eduardo Chapaca Garzón y Jairo David Rojas Bustamante autorizamos a la Universidad Politécnica Salesiana la publicación total o parcial de este trabajo de grado y su reproducción sin fines de lucro.

Además declaramos que los conceptos y análisis desarrollados y las conclusiones del presente trabajo son de exclusiva responsabilidad de los autores.

_____ _____

Jorge Eduardo Chapaca Garzón
CI: 1722373030

Jairo David Rojas Bustamante
CI: 1721671087

DEDICATORIA

Dedico este proyecto de tesis a Dios, a mis padres, a mi hermana, a mi familia y al ser más maravilloso que en vida fue Orlando.

Día a día me han apoyado para poder llegar a esta instancia de mis estudios. Su tenacidad y lucha han hecho de ellos un gran ejemplo a seguir.

Jairo Rojas

Este trabajo está dedicado a Dios, mis padres, mi esposa y mis hijos, ya que han sido un apoyo constante para superar los obstáculos que se presentaron durante la carrera, y que con mucho esfuerzo, sacrificio y dedicación he llegado a culminar mis estudios universitarios.

Para ustedes con mucho amor;

Jorge Chapaca

AGRADECIMIENTO

Agradecemos a la Universidad Politécnica Salesiana, a los profesores por haber aportado en nuestra formación académica y también a todos los compañeros de aula por su valiosa ayuda ya que en los momentos más difíciles nos dieron ánimo y fortaleza para culminar nuestros estudios universitarios.

Gracias a todos;

Jairo Rojas y Jorge Chapaca

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

ÍNDICE DE ANEXOS

RESUMEN

ETCP, prototipo de protocolo de transporte basado en comunicación TCP ha sido elaborado como un protocolo host el cual genera nuevas rutas alternativas de transferencia de datos con el fin de entregar al usuario información segura, confiable y de alta disponibilidad que viaja por la red de manera continua evitando que se tenga que realizar una nueva solicitud para el envío. ETCP es capaz de soportar la llegada de dicha información en desorden armando las partes conforme llegan desde el servidor hacia el cliente, sin tener contra tiempo alguno en el envío y recepción de segmentos.

ABSTRACT

ETCP, prototype-based transport protocol TCP has been prepared as a host protocol which generates new alternative routes data transfer in order to provide the user with secure, reliable and highly available network that travels so continuously avoiding the need to make a new request for shipping. ETCP is able to support the arrival of such information in disarray arming the parties as they arrive from the server to the client without any time have against the sending and receiving segments.

INTRODUCCIÓN

El trabajo se fundamenta en la propuesta de un nuevo prototipo de protocolo de transporte el mismo que contiene toda la documentación referente al proceso de desarrollo del prototipo de protocolo ETCP, se considera desarrollar un nuevo prototipo de protocolo de transporte basado en comunicación TCP que brinde un servicio de excelencia, calidad con capacidad de cubrir las necesidades de transferencia de datos seguros, confiables y de alta disponibilidad, el cual ayude a cumplir con los objetivos propuestos.

A continuación se describe el desarrollo de los capítulos de la investigación:

En el capítulo uno, se plantea la problemática de la investigación, es decir se realiza un análisis de la situación actual de los protocolos y se define cual es la razón de ser de esta tesis. También se describe el objetivo general y específico, enfatizando en el análisis, diseño y desarrollo de un nuevo prototipo de protocolo de transporte basado en comunicación TCP, se describe la hipótesis y las variables del proyecto.

En el capítulo dos, se detalla el marco referencial de la investigación la cual contiene las generalidades investigativas del prototipo de protocolo en referencia a los dos protocolos más importantes que la componen: Protocolo de Control de Transporte (TCP) y Protocolo de Internet (IP), también se detallan conceptos claves que son necesarios para su entendimiento, como son los algoritmos, seguridades y la arquitectura que será utilizada para plantear la solución.

En el capítulo tres, se detalla la metodología del trabajo de investigación para la elaboración del prototipo de protocolo de transporte, se describe el análisis de requerimientos, alcance y la implementación del algoritmo de enrutamiento para la funcionalidad del router virtual básico. Se diseña la estructura y la arquitectura de prototipo, la especificación en comunicación de datos, parámetros de comando, interfaces y tipos de conexión.

En el capítulo cuatro, se detalla la estructura para la elaboración del router virtual, el cual se encuentra desarrollado bajo tecnología Microsoft utilizando Visual Studio .NET 2010 y C#, que utiliza una interfaz que proporciona un entorno gráfico intuitivo para el desarrollo y programación de formularios.

Se analizó con dos interfaces principales, una para el router y otra para los host, como se trata de un prototipo base las interfaces permiten el envío de mensajes entre los distintos puntos de las redes que la conforman, con el objetivo de obtener las siguientes funcionalidades: interfaz de router, interfaz de cliente, Envío de mensajes cliente – router – cliente, parsing de mensajes, con método de posicionamiento, encriptación de mensajes y ruteo.

En el capítulo cinco, se realizan las pruebas experimentales del desarrollo de la aplicación, para lo cual se tomó en cuenta los siguientes aspectos: enrutamiento para la funcionalidad del router virtual básico, confiabilidad, disponibilidad y seguridad. Describe la topología para el caso de estudio de este proyecto. Con las pruebas realizadas se obtienen datos que generan información para generar una lectura acerca del rendimiento del nuevo prototipo de protocolo de transporte. Para la evaluación del prototipo se apoyó en el software wireshark así como de networkactiv el mismo que genera ayuda para la obtención de gráficas estadísticas en la ejecución del sistema.

Además el protocolo TCP y el nuevo prototipo de protocolo tienen un funcionamiento similar en cuanto al uso de los recursos, ya que la ejecución de las pruebas se las realizó con un simulador que implementa en su base el protocolo TCP, por lo que el análisis se centra más en el funcionamiento del nuevo prototipo de protocolo.

CAPÍTULO 1
PROBLEMÁTICA

1.1 Antecedentes

Según la Agencia de Investigación de Proyectos Avanzados (Avanced Research Project Agency) ARPA, en 1969; comenta la evolución del protocolo TCP/IP siempre ha estado muy ligada a la internet, en 1969 ARPA, promovida por el departamento de defensa de los Estados Unidos, desarrolló un proyecto experimental de red conmutada de paquetes al que denominó ARPAnet.

ARPAnet comenzó a ser operativa en 1975, pasando entonces a ser administrada por el ejército de los EEUU. En estas circunstancias se desarrolló el primer conjunto básico de protocolos TCP/IP. Posteriormente en la década de los ochenta, todos los equipos militares conectados a la red adoptan el protocolo TCP/IP y se comienza a implementar también en los sistemas Unix. Poco a poco ARPAnet deja de tener un uso exclusivamente militar y permite que centros de investigación, universidades y empresas se conecten a ésta red. Se habla cada vez con más fuerza de internet y en 1990 ARPAnet deja de existir oficialmente.

En los años sucesivos y hasta nuestros días las redes troncales y los modos de interconexión han aumentado de forma imparable. La red internet parece expandirse sin límite, aunque se mantiene siempre una constante: el protocolo TCP/IP. El gran crecimiento de internet ha logrado que este sea el estándar en todo tipo de aplicaciones telemáticas incluidas las redes locales y corporativas, el cual es conocido como intranet, donde TCP/IP adquiere cada día un mayor protagonismo.

La popularidad del protocolo TCP/IP no se debe tanto a internet, sino a una serie de características que responden a las necesidades actuales de transmisión de datos en todo el mundo, a continuación se describen las más sobresalientes:

- Los estándares del protocolo TCP/IP son abiertos y ampliamente soportados por todo tipo de sistemas, es decir, se puede disponer libremente de ellos y son desarrollados independientemente del hardware de los ordenadores o sistemas operativos.

- TCP/IP funciona prácticamente sobre cualquier tipo de medio, no importa si es una red Ethernet, una conexión ADSL o una fibra óptica.
- TCP/IP emplea un esquema de direccionamiento que asigna a cada equipo conectado a una dirección única en toda la red, aunque esta sea tan extensa como la de internet.

La naturaleza abierta del conjunto de protocolos TCP/IP requiere de algunos estándares y estos están disponibles en documentos de acceso público. Actualmente todos los estándares descritos para este prototipo de protocolo son publicados como RFC (normas, tecnologías, protocolos relacionados con internet, redes en general).

1.2 Justificación

Dada la problemática planteada se ve la necesidad de crear un prototipo de protocolo de transporte capaz de sustentar las necesidades de alta disponibilidad de datos, mejorando las características más relevantes de los protocolos actuales se puede generar un protocolo robusto, fiable y compatible con los existentes.

Este proyecto se considera como una novedad teórica y de alta relevancia práctica, ya que para la creación del nuevo prototipo de protocolo de transporte se estudiará a fondo los existentes, extrayendo las características principales de cada uno para finalmente obtener el funcionamiento de un router virtual básico basado en TCP/IP, con capacidad para adaptar el nuevo protocolo para cubrir las necesidades de transferencia de datos seguros, confiables y de alta disponibilidad.

La obtención de información sobre la problemática, permitirá establecer los ejes sobre los cuales se podrá desarrollar la propuesta antes mencionada. La aplicación de las bases teóricas y experimentales sobre la investigación permitirá alcanzar resultados óptimos, reales y confiables en el área de aplicación de la propuesta.

1.3 Definición del problema

En el mundo actual de las telecomunicaciones la transmisión de datos enviados por la red es cada vez más frecuente por medio de intranet o internet, perder parte de los datos que se envían es un problema que se puede suscitar en la red, generado por una congestión o porque se perdió la ruta que se tenía prevista para el envío de los datos.

Si bien es cierto el tamaño de los paquetes a enviar con el desarrollo de las actuales tecnologías no es problema, sin embargo que se pierda la información al momento de enviar si lo es, debido a esto se genera la necesidad de crear una nueva forma de transportar los datos; de manera que siempre se tenga la información disponible considerando que la transferencia de datos sea segura, confiable y de alta disponibilidad manteniendo una estrategia de búsqueda abierta a cambios.

Por lo que se considera desarrollar la especificación de un nuevo prototipo de protocolo de transporte, para lo cual se toma como base al protocolo TCP. Esta especificación detalla un prototipo con funcionalidad básica para la comunicación y envío de contenidos, de los protocolos y algoritmos de comunicación, seguridad y encriptación actual, se extraerá lo mejor de éstos para buscar la mejor combinación eficiente para la transmisión de contenidos, también se unifica los conceptos y se desarrolla la base para el diseño de una nueva especificación del mismo. Así, se determina el análisis, diseño y desarrollo de un prototipo de protocolo de ruteo basado en comunicación TCP, con capacidad de cubrir las necesidades de transferencia de datos seguros, confiables y de alta disponibilidad.

1.4 Alcance

Se desarrollará la especificación de un nuevo prototipo de protocolo de transporte, tomando en cuenta como base al protocolo TCP. Esta especificación detallará un prototipo con funcionalidad básica para la comunicación y envío de contenidos, para esto se tomará como referencia los protocolos y algoritmos de comunicación, seguridad y encriptación actual, para extraer lo mejor de estos buscando de esta forma la combinación eficiente para la transmisión de contenidos. La especificación cubrirá los siguientes aspectos; envío de datos (comunicación), algoritmos de encriptación (seguridad), algoritmos de búsqueda (disponibilidad), no se cubrirán temas correspondientes a manejo de predicciones o inteligencia artificial.

Los algoritmos necesarios para el prototipo serán desarrollados utilizando la herramienta .NET de microsoft visual studio 2010, como lenguaje de programación se utilizará C#. Al ser tecnología microsoft se garantizará el funcionamiento del software en ambientes windows.

Uno de los aspectos importantes del prototipo de protocolo de transporte será su implementación, para lo cual se desarrollará una aplicación básica que servirá como router virtual básico que hará funcionar el nuevo prototipo. Este tendrá la capacidad operativa de implementación en cualquier máquina con ambiente windows, adicionalmente el software tendrá una bitácora de logs para poder revisar las acciones del router virtual en tiempo real.

Una vez implementado el protocolo, las tareas del router virtual serán las siguientes:

- Compresión y encriptación de los datos a enviar
- Búsqueda de la ruta más corta para llegar a su destino
- Envío de los datos en paquetes
- Enrutamiento de los paquetes
- Recepción de los datos en paquetes
- Segmentación de datos

1.5 Formulación y sistematización

1.5.1 Formulación

La creación de un prototipo de protocolo de transporte, permitirá mejorar él envió de datos a través de los protocolos disponibles cubriendo las necesidades de transferencia de datos seguros, confiables y de alta disponibilidad.

1.5.2 Sistematización

- ¿El proceso de transmisión de datos que manejan los protocolos provee datos seguros, confiables y de alta disponibilidad?

- ¿Existen soluciones en la transmisión de datos cuando hay fallas en las líneas de transmisión y de recepción?

- ¿El nivel que afecta cuando un volumen de información extenso es perdido por medio del prototipo de protocolo de transporte es considerado alto?

- ¿Se ha considerado a la tecnología como herramienta idónea para solventar problemas de pérdida de información en la transmisión de datos a través de la red?

1.6 Objetivos

1.6.1 Objetivo general.

Analizar, diseñar y desarrollar un prototipo de protocolo de transporte basado en comunicación TCP con capacidad de cubrir las necesidades de transferencia de datos seguros, confiables y de alta disponibilidad.

1.6.2 Objetivos específicos

- Realizar un diagnóstico situacional sobre la problemática planteada para el mejoramiento de la transmisión de datos y que éste sea seguro, confiable y de alta disponibilidad por medio del prototipo de protocolo de transporte basado en comunicación TCP.

- Determinar la fundamentación que sustente la propuesta para el mejoramiento de la transmisión de datos con la finalidad de entregar la información segura, confiable y de alta disponibilidad.

- Diseñar un prototipo de protocolo de transporte basado en comunicación TCP, con el fin de entregar al usuario información segura, confiable y de alta disponibilidad.

- Desarrollar algoritmos que permitan la conexión entre cada punto de la red para optimizar y generar rutas de acceso rápidas, confiables y seguras para la transferencia de la información.

- Verificar la funcionalidad del nuevo prototipo de protocolo de transporte ETCP mediante la generación de una aplicación básica en .NET.

- Evaluar los resultados obtenidos de la ejecución de las pruebas para determinar las correspondientes conclusiones y recomendaciones.

1.7 Hipótesis

El desarrollo de un prototipo de protocolo de transporte estará en capacidad de cubrir las necesidades de transferencia de datos seguros, confiables y de alta disponibilidad.

1.8 Variables

1.8.1 Tipos de variables

Variable Dependiente

Prototipo de protocolo de transporte basado en comunicación TCP.

Variable Independiente

- Router virtual capaz de hacer funcionar el nuevo prototipo de protocolo de transporte

Variables Intervinientes

- Evaluación de transferencia de datos seguros
- Análisis de confiabilidad del prototipo de protocolo
- Garantizar alta disponibilidad en la transmisión de datos

CAPÍTULO 2
MARCO REFERENCIAL

2.1 Generalidades

Internet es un conjunto de redes de comunicación interconectadas en la que se encuentran ordenadores con diferentes sistemas operativos, que proporcionan servicios con sus propios conjuntos de protocolos para la comunicación.

Resulta necesario establecer un conjunto de reglas comunes para la comunicación entre estos diferentes elementos con el fin de optimizar la utilización de recursos por muy lejos que se encuentren. Las reglas necesarias para la comunicación entre diferentes equipos de cómputo las tienen los protocolos, las mismas que están presentes en todas las etapas de comunicación, como por ejemplo, la transmisión de flujos de bits a un medio físico hasta aquellas de más alto nivel como el compartir o transferir información desde una computadora a otra en la red.

Este conjunto de protocolos se denomina TCP/IP, en referencia a los dos protocolos más importantes que la componen: protocolo de control de transporte (TCP) y protocolo de internet (IP), que según la RFC fueron dos de los primeros en definirse.

Para el análisis se consideran algunos términos importantes, que se describen a continuación:

- **ACK:** en la comunicación, es un mensaje para confirmar la recepción de uno o varios mensajes que el destino le envía al origen. Si el terminal de destino tiene capacidad para detectar errores, el ACK debe informar que el mensaje se ha recibido de forma íntegra.
- **SYN:** es un bit de control localizado dentro del segmento TCP el cual indica la generación de un pedido para establecer una conexión.
- **Sincronización:** se utiliza para regresar del estado actual a un estado anterior conocido como error durante la sesión. La capa de transporte sólo recupera errores de comunicación los cuales generan en el nivel de sesión entre usuarios (capas superiores).

- **FIN:** es un indicador de datos en TCP, si se encuentra fijado en uno automáticamente se interrumpe la conexión.
- **ICMP:** es un protocolo el cual permite administrar información relacionado con errores de los equipos en red; éste no permite corregir errores sino pasa una notificación a los protocolos de las capas cercanas.
- **HDLC**: es un protocolo de comunicación de datos punto a punto el cual opera a nivel de enlace de datos y ofrece una comunicación confiable entre el transmisor y el receptor; permite proporcionar la recuperación de errores.
- **RFC:** son series de notas y sistemas que se conectan a internet. Cada nota es un documento cuyo contenido son una propuesta oficial para un nuevo protocolo de la red internet (originalmente de ARPANET), en el cual se explica detalladamente en caso de ser aceptado y proceder a su implementación sin ambigüedades. Los protocolos en su mayoría han sido diseñados e implementados por investigadores y científicos, los cuales han sido expuestos en forma de RFC.

2.1.1 SUITE TCP/IP

En 1972 TCP/IP fue desarrollado y demostrado por primera vez por el departamento de defensa de los Estados Unidos, ejecutándolo en ARPANET, una red de área extensa de dicho departamento. Los distintos protocolos de la suite TCP/IP hacen posible acceder a diferentes servicios de red, éstos protocolos trabajan para proporcionar el transporte de datos dentro de internet o intranet. Servicios como: Transmisión de correo electrónico, noticias, acceso a la world wide web (WWW); entre otros. TCP/IP posee ventajas respecto a otros protocolos; por ejemplo, se lo puede implementar a un bajo costo y consumir pocos recursos de red.

En 1983, TCP/IP se integró en la versión 4.2 del sistema operativo UNIX de berkeley y la integración en versiones comerciales de UNIX vino pronto por el avance tecnológico; así es como TCP/IP se convirtió en el estándar de internet.

Actualmente TCP/IP no solo se usa para la internet; por ejemplo: A menudo se diseñan intranets usando TCP/IP. En estos entornos, TCP/IP ofrece ventajas sobre otros protocolos de red; una de ellas es que trabaja sobre una gran variedad de hardware y sistemas operativos, dicha red puede contener estaciones Mac, PC

compatibles, estaciones SUN, servidores Novell, etc. Todos éstos elementos pueden comunicarse usando la misma suite de protocolos TCP/IP.

La tabla que se detalla a continuación indica una lista de plataformas que soportan TCP/IP:

Tabla 1: Lista de plataformas que soportan TCP/IP

Plataforma	Soporte de TCP/IP
Unix	Nativo
Dos	Piper/ IP por Ipswitch
Windows	TCP man por Trumpet Software
Windows 95	Nativo
Windows NT	Nativo
Macintosh	Mac TCP u OpenTransport(Sys 7.5+)
Os/2	Nativo
As /400 Os/400	Nativo

Elaborado por: Jorge Chapaca y David Rojas

2.1.2 Modelo OSI analizado desde TCP/IP

TCP/IP opera a través del uso de capas, las mismas que son necesarias para completar una transferencia de datos entre dos máquinas. Las capas están divididas como se ilustra en la tabla 2.

Tabla 2: Capas de la pila a través de la cual opera TCP/IP

EQUIPO SERVIDOR O CLIENTE	
Capa de Aplicaciones	Cuando un usuario inicia una transferencia de datos, esta capa pasa la solicitud a la Capa de transporte.
Capa de Transporte	Esta capa añade una cabecera y pasa los datos a la Capa de Red.
Capa de Red	En la Capa de Red, se añade las direcciones IP de origen y destino para el enrutamiento de datos.
Capa de Enlace de Datos	Ejecuta un control de errores sobre el flujo de datos entre los protocolos anteriores y la Capa Física.
Capa Física	Ingresa o egresa los datos a través del medio físico, que puede ser Ethernet vía coaxial, PPP vía modem, etc.

Elaborado por: Jorge Chapaca y David Rojas

Cada capa está asociada con múltiples protocolos que trabajan sobre los datos; así como también pueden enviar y recibir datos desde la capa adyacente. Una vez que los datos han pasado a través del proceso TCP/IP viajan a su destino en otra máquina de la red y el proceso se ejecuta nuevamente, pero al revés (los datos entran por la capa física y recorren la pila hacia arriba).

2.2 Protocolos de transporte

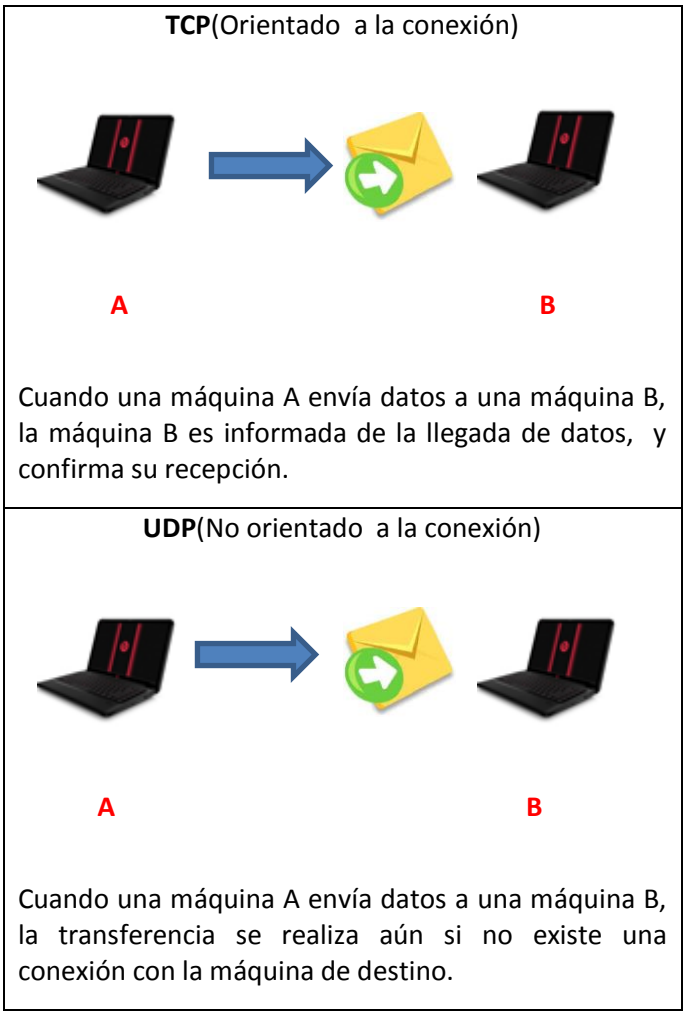
En el cuarto nivel de las capas del modelo OSI, se encuentra la capa de transporte la misma que se encarga de la primera transferencia de datos libres de errores entre el emisor y el receptor aunque no estén directamente conectados, así como de mantener el flujo en la red. La principal tarea de esta capa es la de brindar el medio necesario para transportar los mensajes que se envían entre dos puntos de la red de manera segura y confiable.

El objetivo primordial de los protocolos en la capa de transporte reside en garantizar la comunicación fiable, proporcionando un servicio eficiente de "extremo a extremo".

Tipos de servicio:

- Servicio orientado a conexión: TCP
- Servicio sin conexión: UDP

Figura 1: Servicios de los protocolos de transporte



Elaborado por: Jorge Chapaca y David Rojas

Principales Funciones:

- Segmentación de los datos y de su posterior re ensamblado
- Supervisión de la llegada en orden de los segmentos

- Controla los errores y el flujo, el receptor informa al emisor del espacio de almacenamiento disponible. El emisor no debe enviar más información de la que se le ha permitido

Estructura de mensajes:

La estructura de los mensajes del control de protocolos hasta este nivel se compone de:

- Un identificador de host por medio de su dirección IP
- Un identificador de proceso por medio de la asignación de un puerto
- Un identificador del protocolo que se está utilizando: TCP o UDP

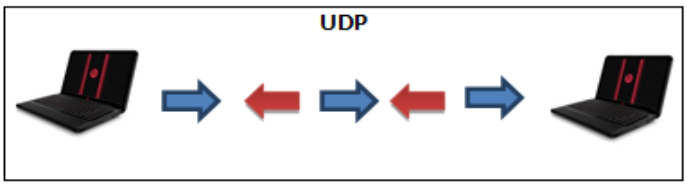
2.2.1 UDP (User Datagram Protocol):

Es un protocolo de transporte no orientado a la conexión, que proporciona un servicio de "datagramas de usuario" que se caracteriza por no incluir mecanismos que eviten la pérdida de mensajes, es decir, no ofrece fiabilidad, lo cual implica que las aplicaciones que lo utilicen deben responsabilizarse de este tipo de problema. El mismo está basado en el envío y recepción de datagramas.

Este permite el envío de datagramas por la red sin que se hayan establecido las debidas conexiones antes, debido a que toda la información necesaria para la comunicación esta embebida dentro del mismo.

No posee ningún tipo de control de flujo por lo que los segmentos enviados no necesariamente llegan en el orden en el que son enviados, de tal forma no es seguro que el envío haya llegado al destino.

Figura 2: Comunicación a través de UDP



Elaborado por: Jorge Chapaca y David Rojas

2.2.1.1 Formato del datagrama UDP:

El formato del datagrama UPD se basa en el campo longitud mensaje que incluye a la cabecera y a los datos del mismo. El campo checksum es opcional, cuando no se emplea se pone a cero.

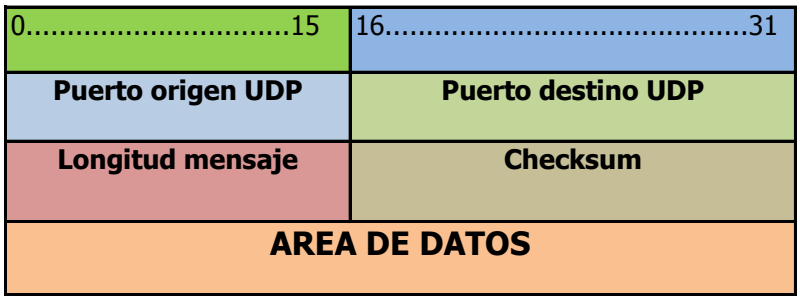
Puerto origen: Identifica al puerto del proceso de aplicación remitente. Es opcional, si no se utiliza se pone en cero.

Puerto destino: Identifica el proceso de recepción en el ordenador de destino.

Longitud mensaje: Indica la longitud del datagrama de usuario, incluyendo la cabecera y los datos. La longitud mínima es de 8 octetos.

Checksum: Contiene el valor del complemento a 1 en 16 bits del complemento a 1 de la suma de la pseudo-cabecera de IP, la cabecera de UDP y los datos.

Figura 3: Formato datagrama UDP



Elaborado por: Jorge Chapaca y David Rojas

2.2.2 Protocolo de Control de Transmisión (TCP)

Es uno de los principales protocolos de la capa de transporte del modelo TCP/IP.
TCP es un protocolo orientado a conexión, es decir, permite que dos máquinas que están comunicadas controlen el estado de la transmisión.

2.2.2.1 Principales Características

- **Orientado a conexión:** esto quiere decir que TCP mantiene información del estado de cada cadena de datos que circula por él.

- **Conexión de inicio confiable:** garantiza una conexión de inicio confiable y sincronizado entre los dos extremos de la conexión.
- **Conexión de finalización aceptable:** TCP garantiza la entrega de todos los datos antes de la finalización de la conexión.
- **Resolver de forma automática:** resuelve los problemas que se puedan dar durante el intercambio de información como los fallos en los enlaces, errores, pérdidas o duplicación de datos, entre otros.

2.2.2.2 Funciones Principales

- **Operación Full-Duplex**: una conexión TCP son un par de circuitos virtuales, cada uno en una dirección. Sólo los dos sistemas finales sincronizados pueden usar la conexión.
- **Error Checking**: es una técnica de checksum usada para verificar que los segmentos no estén corrompidos.
- **Acknowledgements**: recibo de uno o más segmentos, el receptor regresa un acknoledgement (reconocimiento) al transmisor indicando que recibió los paquetes. Si los paquetes no son notificados, el transmisor puede reenviar los segmentos o terminar la conexión si el transmisor cree que el receptor no está más en la conexión.
- **Flow Control**: control del envío de datos cuando el receptor no tiene la capacidad de leer los segmentos tan rápido como se le envía, el receptor descarta segmentos. Los segmentos fallidos alertan al receptor para bajar la tasa de transferencia o dejar de transmitir.
- **Servicio de recuperación de segmentos**: el receptor puede solicitar la retransmisión de un segmento. Si el segmento no es notificado como recibido.

Con el uso del protocolo TCP, las aplicaciones se pueden comunicar en forma segura sin tener en cuenta las capas que intervengan. Esto significa que los routers que funcionan en la capa de internet solo tienen que enviar los datos en forma de datagramas, sin preocuparse con el monitoreo de datos porque esta función la cumple la capa de transporte, es decir específicamente el protocolo TCP, lo que implica una reducción en la complejidad de los algoritmos de los routers.

En TCP para que una comunicación ocurra es necesario que las máquinas que lo requieren establezcan primero una conexión. La máquina emisora que solicita la conexión se llama cliente, y la máquina receptora se llama servidor, este tipo de comunicación toma el nombre de entorno cliente-servidor. Las máquinas de dicho entorno se comunican en modo en línea, es decir, que la comunicación se realiza en ambas direcciones en tiempo real.

Otra función del TCP es la capacidad de controlar la velocidad de los datos para emitir mensajes de tamaño variable. Estos mensajes se llaman segmentos.

2.2.2.3 Multiplexión en TCP

TCP posibilita la realización de una tarea importante: multiplexar/demultiplexar; la cual transmite los datos desde diversas aplicaciones en la misma línea. Estas operaciones se realizan empleando el concepto de puertos o conexiones, y quiere decir que un número vinculado a un tipo de aplicación cuando se combina con una dirección IP determina en forma exclusiva una aplicación que se ejecuta en una máquina determinada., (es.kioskea.net, 2011)

2.2.2.4 Confiabilidad de las transferencias

El protocolo TCP permite garantizar la transferencia de datos confiable, a pesar de que usa el protocolo IP que no incluye ningún monitoreo de la entrega de datagramas. De hecho, el protocolo TCP tiene un sistema de acuse de recibo que permite al cliente y al servidor garantizar la recepción mutua de datos. Cuando se emite un segmento se lo vincula a un número de secuencia.

Con la recepción de un segmento de datos, la máquina receptora devolverá un segmento de datos donde el indicador ACK esté fijado en 1 para poder indicar que es un acuse de recibo acompañado por un número de acuse de recibo que equivale al número de secuencia anterior. Además, usando un temporizador que comienza con la recepción del segmento en el nivel de la máquina originadora, el segmento se reenvía cuando ha transcurrido el tiempo permitido, ya que en este caso la máquina originadora considera que el segmento está perdido. Sin embargo, si el segmento no está perdido y llega a

su destino, la máquina receptora lo sabrá, gracias al número de secuencia que es un duplicado y solo retendrá el último segmento que llegó al destino., (es.kioskea.net, 2011)

2.2.2.5 Establecimiento de una conexión

Este proceso de comunicación se produce con la transmisión y recepción de datos, que se basa en un número de secuencia, la máquina originadora y receptora (cliente y servidor) deben conocer el número de secuencia inicial de la otra máquina.

La conexión establecida entre las dos aplicaciones se realiza a través de los siguientes pasos:

- Los puertos TCP deben estar abiertos.
- La aplicación en el servidor es pasiva, ya que la aplicación escucha y espera una conexión.
- La aplicación del cliente realiza un pedido de conexión al servidor pasivo. La aplicación del cliente se considera "abierta activa".

Este diálogo posibilita el inicio de la comunicación y se lo realiza en tres etapas, que se describe a continuación:

a) **En la primera etapa,** el transmisor envía un segmento donde el indicador SYN está fijado en 1 (para indicar que es un segmento de sincronización), con número de secuencia N llamado número de secuencia inicial del cliente.
b) **En la segunda etapa,** el receptor toma el segmento inicial que viene del cliente y luego le envía un acuse de recibo, que es un segmento en el que el indicador ACK está fijado en 1 y el indicador SYN está fijado en 1, ya que se considera una nueva sincronización. Este segmento incluye el número de secuencia de esta máquina (el receptor), que es el número de secuencia inicial para el transmisor. El campo más importante en éste segmento es el de acuse de recibo que contiene el número de secuencia inicial del transmisor incrementado en 1.

c) **En la última etapa,** el transmisor envía un acuse de recibo, que es un segmento donde el indicador ACK está fijado en 1 y el indicador SYN está fijado en 0 ya que no es un segmento de sincronización; su número de secuencia está incrementado y el acuse de recibo representa el número de secuencia inicial del servidor incrementado en 1., (es.kioskea.net, 2011)

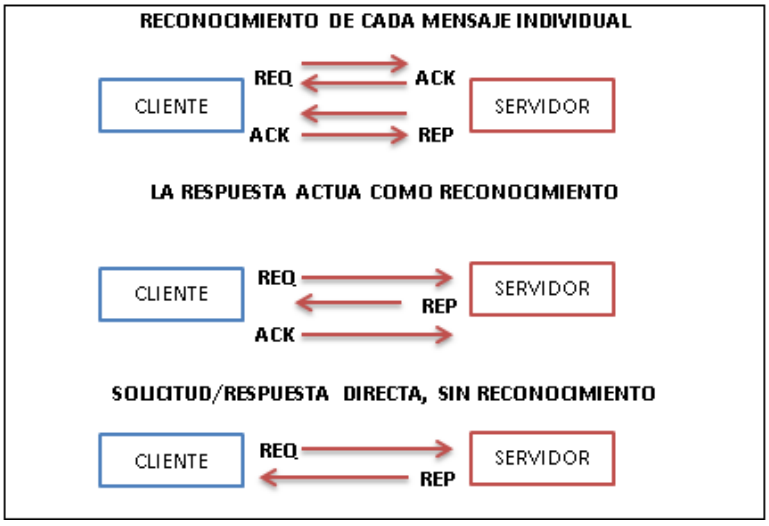
Después de esta secuencia con tres intercambios, las dos máquinas están sincronizadas y la comunicación puede comenzar.

2.2.2.6 Finalización de una conexión

El cliente puede pedir que se termine una conexión del mismo modo que el servidor a través del siguiente procedimiento:

- Una de las máquinas envía un segmento con el indicador FIN fijado en 1, y la aplicación se auto coloca en estado de espera, es decir que deja de recibir el segmento actual e ignora los siguientes.
- Una vez receptado este segmento, la otra máquina envía un acuse de recibo con el indicador FIN fijado en 1 y sigue enviando los segmentos en curso; después de esto, la máquina informa a la aplicación que se ha recibido un segmento FIN y luego envía un segmento FIN a la otra máquina, que cierra la conexión.

Figura 4: Establecimiento y Finalización de una conexión



Elaborado por: Jorge Chapaca y David Rojas

Es necesario tener en cuenta cuando termina una sesión ya que esto permite al servidor liberar recursos que no están siendo usados y poder alojar nuevas conexiones, este esquema adicionalmente permite que el servidor deje de tratar de enviar información a puntos muertos de la red.

2.2.2.7 Modelo de estratificación por capas de TCP/IP

El segundo modelo de mayor estratificación es TCP/IP, el cual está organizado en cuatro capas conceptuales que se construyen sobre una quinta capa de hardware.

El esquema detallado a continuación muestra las capas conceptuales así como la forma en que los datos pasan entre ellas.

Tabla 3: Capas conceptuales paso de objetos entre capas

CAPAS
1. Aplicación
2. Transporte
3. Internet
4. Interfaz de Red

Elaborado por: Jorge Chapaca y David Rojas

- **Capa de Aplicación**

Es el nivel más alto, los usuarios desean tener una aplicación que acceda a servicios disponibles a través de la red. Una aplicación interactúa con uno de los protocolos de nivel de transporte para enviar o recibir datos. Cada programa de aplicación selecciona el tipo de transporte necesario, el cual puede ser una secuencia de mensajes individuales o un flujo continuo de octetos. El programa de aplicación pasa los datos en la forma requerida hacia el nivel de transporte para su entrega.

- **Capa de Transporte**

La capa de transporte es muy necesaria porque se encarga de generar la comunicación entre host origen y destino, teniendo como responsabilidad la segmentación de los datos en el origen y así facilita su transferencia a las capas

inferiores. La segmentación sirve para evitar que la comunicación entre host se limite a una sola función ya que en la red existen muchos usuarios que la están utilizando, la segmentación da lugar a la multiplexión y esto permite múltiples sesiones conectadas a la misma red enviando o recibiendo datos al mismo tiempo. El host destino controla el orden de los segmentos y el flujo de datos que se reciben y de esta manera se evita el agotamiento de recursos. Lo siguiente es mandar un mensaje al host que está enviando los datos para que disminuya la velocidad de transferencia y si existe un error en un segmento pedir el reenvío de este, la capa de transporte también se encarga de ensamblar los segmentos de forma ordenada para que la aplicación adecuada los muestre a los usuarios.

Cabe recordar que la función principal de la capa de transporte es administrar los datos de aplicaciones para las conversaciones entre hosts. Los dos protocolos más comunes de la capa de transporte del conjunto de protocolos TCP/IP son el protocolo de control de transmisión (TCP) y el protocolos de datagramas de usuario (UDP). Ambos protocolos gestionan la comunicación de múltiples aplicaciones. Las diferencias entre ellos son las funciones específicas que cada uno implementa., (edpucese.blogspot.com, 2013)

- **Capa de Internet**

El propósito de la capa de internet es el enviar paquetes origen desde cualquier punto de la red y que estos paquetes lleguen a su destino independientemente de la ruta y de las redes circulan hasta llegar al destino, en esta capa se produce la determinación de la mejor ruta y la conmutación de paquetes. El protocolo específico que rige esta capa se denomina protocolo de internet (IP), el mismo proporciona los servicios básicos de transmisión de paquetes sobre los cuales se construyen todas las redes TCP/IP., (es.scribd.com, 2012)

La capa Internet también maneja la entrada de datagramas, verifica su validez y utiliza un algoritmo de ruteo para decidir si el datagrama debe procesarse de manera local o debe ser transmitido. Para el caso de los datagramas direccionados hacia la máquina local, el software de la capa de red de redes borra el encabezado del datagrama y selecciona, de entre varios protocolos de transporte, un protocolo con el que manejará el paquete. Por último, la capa Internet envía los mensajes

ICMP de error y control necesarios y maneja todos los mensajes ICMP entrantes.”, (docente.ucol.mx, 2011)

- **Capa de Interfaz de red**

Este nivel se limita a recibir datagramas del nivel superior (nivel de red) y transmitirlo al hardware de la red. El software TCP/IP de nivel inferior consta de una capa de interfaz de red responsable de aceptar los datagramas IP y transmitirlos hacia una red específica.”, (docente.ucol.mx, 2011)

2.2.2.8 Modelo de estratificación por capas de OSI

Interconexión de sistemas abiertos - OSI; es el nombre de una arquitectura de capas para redes de ordenadores y sistemas distribuidos propuesto como estándar de interconexión de sistemas abiertos.

- **Capas conceptuales**

El concepto de capas ayuda a comprender la acción que se produce durante el proceso de comunicación de un computador a otro. Existen muchas capas que ayudan a describir los detalles del proceso de flujo.

Figura 5: Capas conceptuales pasó de objetos entre capas

Elaborado por: Jorge Chapaca y David Rojas

- **Capa de aplicación**

En esta capa contiene las interfaces de usuario, aquí es donde los datos son enviados y recibidos por los usuarios. Las peticiones se realizan por las aplicaciones de acuerdo a los protocolos utilizados; así como la capa física, que está en el borde del modelo, por lo que también se inicia y se detiene todo el proceso.

Función: es realizar la interfaz entre los usuarios finales y los programas de comunicación.

- **Capa de presentación**

La capa de presentación tiene la funcionalidad de mostrar el formato de los datos, y permite la representación de ellos. Éste formato incluye la compresión y cifrado de datos.

Es más fácil entender esta capa como la que traduce los datos en un formato para que pueda entender el protocolo que la vaya a implementar. Por ejemplo cuando el transmisor utiliza un estándar diferente de otros a continuación, ASCII, convierte los datos para que sean legibles y entendibles. Cuando dos redes diferentes necesitan comunicarse, es la capa de presentación que funciona, traduce capa de datos de cada uno.

En cuanto a la compresión, se puede entender como un archivador de ficheros - ZIP, RAR - donde el transmisor comprime los datos, descomprime el receptor, y cuando hay necesidad de una mayor seguridad, esta capa aplica esquemas de cifrados.

Función: la encriptación, compresión, formato y la presentación de formatos de datos (por ejemplo, JPEG, GIF, MPEG) para las aplicaciones.

Protocolos: SSL, TLS.

Dispositivos: gateways (protocolos de traducción entre diferentes redes), transmisor-receptor (traducción entre las señales ópticas y eléctricas - que se desplaza en cables diferentes).

- **Capa de sesión**

Esta capa se encarga de iniciar y finalizar la sesión, siendo responsable de la comunicación e intercambio de datos entre las aplicaciones de los sistemas finales. La capa de sesión permite a los usuarios de máquinas diferentes establecer sesiones entre ellos. Una sesión permite el transporte ordinario de datos, como lo hace la capa de transporte, pero también proporciona servicios mejorados que son útiles en algunas aplicaciones.

Función: Iniciar, gestionar y terminar sesiones de la capa de presentación; por ejemplo: sesiones TCP.

- **Capa de transporte**

La capa de transporte debe garantizar la calidad en la entrega y recepción de datos, a su vez, como en todo el transporte, debe ser administrado para ello debe contar con un servicio de calidad. En términos simples, las normas y acciones destinadas a garantizar la calidad de servicio deseado, basado en la recuperación de errores y control de los flujos de datos.

Función: transportar, entrega y recepción de datos de la red, con calidad de servicio.

Protocolos: protocolo de control de transmisión (TCP), protocolo de datagrama de usuario (UDP), intercambio de paquetes secuenciados (SPX).

- **Capa de red**

Es responsable del tráfico de datos. Para ello, cuenta con dispositivos que identifican el mejor camino posible a seguir y establecen dichas rutas.

Esta capa tiene la dirección física MAC (nivel 2-Link) y la convierte en la dirección lógica (dirección IP).

Función: Direccionamiento, enrutamiento y definir las mejores rutas posibles.

Protocolos: ICMP, IP, IPX, ARP, IPSEC.

Dispositivos: Routers.

PDU: Paquetes.

- **Capa de enlace de datos**

La capa de enlace de datos recibe el formato de la capa física, bits, y trata de convertir los datos para que se pueda remitir a la siguiente capa y sea entendible para ésta.

Función: Enlace de datos de un host a otro, se realiza a través de los protocolos definidos para cada medio específico por donde se envían los datos.

Protocolos: PPP, ethernet, FDDI, ATM, token ring.

Dispositivos: Interruptores, tarjeta de red, interfaces.

- **Capa física**

Es la encargada de realizar las conexiones globales de la computadora hacia la red, referente al medio físico como a la forma de transmisión de la información.

Las principales funciones son:

- Definir las características funcionales de la interfaz
- Transmitir el flujo de bits a través del medio
- Manejar las señales eléctricas del medio de transmisión
- Garantizar la conexión
- Definir el medio físico por el cual va a viajar la comunicación

2.3 Algoritmos de enrutamiento

La función principal de la capa de red es la de enrutar paquetes desde la máquina de origen a la máquina de destino. Es aquella parte que se encarga y decide la línea de salida por la que se transmitirá un paquete de entrada. Si la subred usa datagramas de manera interna, esta decisión se tomará cada vez que llegue un paquete de datos, debido que la ruta seleccionada podría haber cambiado desde la última vez.

> Si la subred usa circuitos virtuales internamente, las decisiones de enrutamiento se toman sólo al establecer un nuevo circuito virtual. Los paquetes de datos simplemente siguen la ruta previamente establecida sucesivamente; este caso es denominado como enrutamiento de sesión, dado que una ruta permanece vigente durante toda la sesión de usuario, por ejemplo; durante una sesión desde una terminal o durante una transferencia de archivos., (Tanenbaum, 2003, pág. 350)

> En algunas ocasiones se debe distinguir entre el enrutamiento que es el proceso de la toma de decisión de ruta a utilizar y el reenvío el cual consiste en la acción que se toma cuando llega un paquete. Se puede considerar que un enrutador realiza dos procesos internos, uno de ellos maneja cada paquete con prioridad de llegada, buscando en las tablas de enrutamiento la línea de salida por la cual se enviará, dicho proceso se conoce como reenvío. El otro proceso es responsable de llenar y actualizar las tablas de enrutamiento, es ahí donde entra en acción el algoritmo de enrutamiento., (Tanenbaum, 2003, pág. 351)

> Los algoritmos de enrutamiento pueden agruparse en dos clases principales: adaptativos y no adaptativos: Los algoritmos adaptativos cambian sus decisiones de enrutamiento para reflejar los cambios de topología y por lo general también el tráfico. Los algoritmos adaptativos difieren en el lugar de donde obtienen su información, es decir localmente de los enrutadores adyacentes o de todos los enrutadores, el momento de cambio de sus rutas, por ejemplo, cada Δt segundos, cuando cambia la carga o cuando cambia la topología y la métrica usada para la optimización es decir la distancia, número de saltos o tiempo estimado de tránsito. Y los algoritmos no adaptativos no basan sus decisiones de enrutamiento en mediciones o estimaciones del tráfico y la topología actual. La decisión de qué ruta a utilizar para llegar de una máquina a otra se toma por adelantado fuera de línea y se carga en los enrutadores al arrancar la red. Éste procedimiento se denomina enrutamiento estático.”, (Tanenbaum, 2003, pág. 352)

A continuación se detalla una serie de algoritmos de enrutamiento estáticos y dinámicos, los cuales son:

- **Enrutamiento por la ruta más corta**

Es un algoritmo de enrutamiento estático. "primero la ruta libre más corta" (OSPF – open shortest path first), recrea la topología exacta de toda la red. Su métrica se basa en el retardo, ancho de banda, carga y confiabilidad de los distintos enlaces posibles para llegar a un destino en base a esos conceptos, el protocolo prefiere una ruta por sobre otra.

Estos protocolos utilizan un tipo de publicaciones denominadas publicaciones de estado de enlace (LSA) que intercambian entre los routers mediante esta publicación, cada router crea una base de datos de la "topología de la red completa, las cuales son:

- Buscar una unión común de la topología de la red
- Cada dispositivo calcula la ruta más corta a los otros routers
- Activar las actualizaciones de los eventos de la red
- Transmitir", (González, 2011, pág. 210)

- **Algoritmo por Inundación**

La inundación es un algoritmo estático, y consiste en que cada paquete de entrada se envía por cada una de las líneas de salida, excepto aquella por la que llegó. La inundación evidentemente genera grandes cantidades de paquetes duplicados, cantidades infinitas por lo que se toma medidas para limitar este proceso. Una de las medidas a utilizar es un contador de escalas contenido en la cabecera de cada paquete, y permite disminuir en cada escala descartándose al llegar el contador a cero. El contador debe inicializarse a la longitud de la trayectoria, inicializar el contador en el peor de los casos; es decir el diámetro de la subred, una variación de la inundación más práctica es la inundación selectiva; en este algoritmo los enrutadores no envían cada paquete de entrada por todas las líneas sino sólo por aquellas que van aproximadamente en la dirección correcta.

La inundación no es práctica en la mayoría de las aplicaciones, pero tiene algunos usos como por ejemplo: "en aplicaciones militares y en las aplicaciones de bases de datos distribuidas es necesario actualizarlas continuamente, solo en estos casos puede ser útil este algoritmo.", (exa.unne.edu.ar, 2002, pág. 5)

- **Algoritmo Vector Distancia**

Es un algoritmo de enrutamiento dinámico, el que determina la dirección y la distancia hacia cualquier enlace de la red.

Su métrica se basa en "número de saltos" nombre denominado en redes, es la cantidad de routers por los que tiene que pasar el paquete para llegar a la red de destino, la ruta que tenga el menor número de saltos es la más óptima y la que se publicará, características que se describe a continuación:

- Visualizar la red desde la perspectiva de los vecinos
- Actualizar periódicamente
- Transmitir copias parciales o completas de las tablas de enrutamiento
- Convergencia lenta
- Incrementar las métricas a través de las actualizaciones
- Trabajar conjuntamente con el algoritmo de Bellman-Ford
- Utilizar con redes que tengan enlaces con coste negativo
- Detectar la existencia de un ciclo negativo en la fuente, (González, 2011, pág. 1)

- **Algoritmo Estado de enlace**

Es un algoritmo de enrutamiento dinámico, se basa en el enrutamiento por estado enlace, es sencillo y se desarrolla en cinco partes, cada enrutador debe realizar varias actividades las cuales se describen a continuación:

- Descubrir a sus vecinos y conocer sus direcciones de red
- Medir el retardo o costo para cada uno de sus vecinos
- Construir un paquete que indique todo lo que acaba de aprender
- Enviar el paquete a todos los demás enrutadores
- Calcular la ruta más corta a todos los demás enrutadores

Toda la topología y los retardos se miden empíricamente que se distribuyen a cada enrutador. Por tal motivo se puede usar el algoritmo de Dijkstra, ya que sirve para encontrar la ruta más corta a los demás enrutadores.

2.4 Algoritmos de búsqueda y comunicación

- **Búsqueda del camino más corto**

El objetivo principal es encontrar el camino con un coste mínimo entre una fuente origen (o) y un destino (d).

Si (o, d) se mantiene constante para todos los enlaces, la solución es la ruta de menor número de saltos.

Algoritmos con una única fuente: se encuentra el camino más corto entre el resto de nodos:

- Dijkstra
- Bellman-Ford

Algoritmos para toda la red: se encuentra el camino más corto entre todas las posibles parejas de nodos en la red:

- Floyd-Warshall
- Johnson

Algoritmo de Dijkstra

El algoritmo consiste en encontrar la ruta más corta entre un punto A y B, así: Dado un grafo, cuyo paso de vértice a vértice ha sido asociado con un coste determinado, se define como camino de coste mínimo de un punto A hacia B, como el camino donde la sumatoria de los costes de los arcos que lo forman es la menor entre todos los caminos posibles. El fundamento en el que se basa el algoritmo es el de optimización: Si el camino más corto entro en los vértices u y v pasa por el vértice w, entonces la parte del camino que va de w a v debe ser el camino más corto entre todos los caminos posibles.

Algoritmo de Bellman-Ford

El algoritmo de Bellman-Ford determina la ruta más corta desde un nodo origen hacia los demás nodos para ello es requerido como entrada un grafo cuyas aristas posean costes. La diferencia de este algoritmo con los demás es que los costes pueden tener valores negativos ya que permite detectar la

existencia de un ciclo negativo para su posterior solución., (jarias.wordpress.com, 2010)

Algoritmo de Johnson

Este algoritmo se encarga de generar el camino más corto en un grafo, teniendo en cuenta que no importan si los costes entre aristas son negativos o positivos. Se puede decir que es un procedimiento más general que el algoritmo de Dijkstra, ya que permite la inclusión de valores negativos en los costes. Se debe tener en cuenta que si el coste total de pasar de un nodo a otro, después de aplicar el procedimiento, es un coste negativo; entonces el grafo no tiene solución; conclusión que el algoritmo tiene la capacidad de deducir.

- **Reenvío de paquetes (Forwarding)**

Cuando un paquete llega al enlace de entrada de un router, éste tiene que pasar el paquete al enlace de salida apropiado. Una característica importante de los routers es que no difunden tráfico broadcast.

- **Enrutamiento de paquetes (Routing)**

Mediante el uso de algoritmos de enrutamiento tiene que ser capaz de determinar la ruta que deben seguir los paquetes a medida que fluyen de un emisor a un receptor.

El objetivo de un router virtual es simular todas las operaciones que el router físico las haría, esto se realiza mediante la simulación por software de las funciones del dispositivo físico, es decir se toman los datos de entrada se aplican los algoritmos que aplicaría el router físico y se devuelve una salida a la red.

2.5 Algoritmos de encriptamiento

Encriptación es un proceso que toma la información, se la modifica para que solo pueda ser interpretada por un esquema que conozca el proceso donde la información fue escondida. Para poder establecer este esquema es necesario que tanto el que envía como el que recibe conozcan el procedimiento de escondido e inverso, ya que de otra manera sería imposible interpretar la información encriptada.

- **Encriptación a nivel de enlace**

Es la forma de protección criptográfica más transparente para los controladores de los dispositivos y para las aplicaciones. Esta protección solo afecta a un enlace individual. La ventaja principal es que el paquete es encriptado por completo, incluyendo las direcciones de origen y destino lo que deja fuera de riesgo la comunicación. El problema es que solo protege un enlace en particular; si un mensaje debe atravesar más de un enlace será vulnerable en el nodo intermedio y en el siguiente enlace, si éste no está protegido, por lo tanto el encriptado a nivel de enlace es útil para proteger solo tráfico local o unas pocas líneas de enlace muy vulnerables o críticas; como por ejemplo circuitos satelitales.", (textoscientificos.com, 2006)

- **Encriptación a nivel de transporte**

EL protocolo de seguridad de la capa de transporte (transport layer security protocol - TLSP); permite a los sistemas comunicarse de forma segura sobre Internet estos son transparentes para la mayor parte de las aplicaciones.", (textoscientificos.com, 2006)

El protocolo está basado en el concepto de id de clave o key-id; la cual es transmitida sin encriptar junto con el paquete encriptado. Permite controlar el comportamiento de los mecanismos de encriptado y desencriptado, especifica el algoritmo y el tamaño del bloque de encriptado, el mecanismo de control de integridad usado el período de validez de la clave, etc. Utiliza un mecanismo de administración para intercambiar claves e id's., (textoscientificos.com, 2006)

El protocolo de seguridad de la capa de transporte-TLSP; está limitado a conexiones individuales tales como circuitos virtuales creados en TCP. Diferentes circuitos entre el mismo par de hosts pueden ser protegidos con diferentes claves. El segmento TCP completo incluyendo el encabezado es encriptado. Este nuevo segmento es enviado al protocolo IP, con un identificado de protocolo diferente. Al recibir el paquete IP envía el paquete a TLSP que luego de desencriptar y verificar el paquete lo pasa a TCP., (textoscientificos.com, 2006)

Estos protocolos no exigen ninguna restricción de comunicación es decir, cualquier host protegido puede comunicarse con cualquier otro. Los patrones de comunicación son una cuestión administrativa, estas decisiones son aplicadas por los sistemas de encriptado y los mecanismos de distribución de claves.", (textoscientificos.com, 2006)

Algoritmo Rijndael

"También llamado AES es un sistema de cifrado por bloques, diseñado para manejar longitudes de clave y de bloque variables, ambas comprendidas entre los 128 y los 256 bits.", (Muñoz, 2004, pág. 19)

La estructura del algoritmo Rijndael está formada por un conjunto de rondas, que no es otra cosa que reiteraciones de 4 funciones matemáticas diferentes. "Por tanto, el algoritmo se basa en aplicar un número de rondas determinado a la información (texto plano) de esta manera se obtiene una información cifrada. La información generada por cada función es un resultado intermedio, que se conoce como estado"., (Muñoz, 2004, pág. 29)

"El algoritmo representa el estado como una matriz rectangular de bytes, que posee 4 filas y Nb columnas. Siendo el número de columnas Nb en función del tamaño del bloque:", (Muñoz, 2004, pág. 31)

Nb = tamaño del bloque utilizado en bits / 32.

La clave del sistema se representa mediante una matriz rectangular de bytes de 4 filas y Nk columnas. Siendo el número de columnas Nk en función del tamaño de la clave:

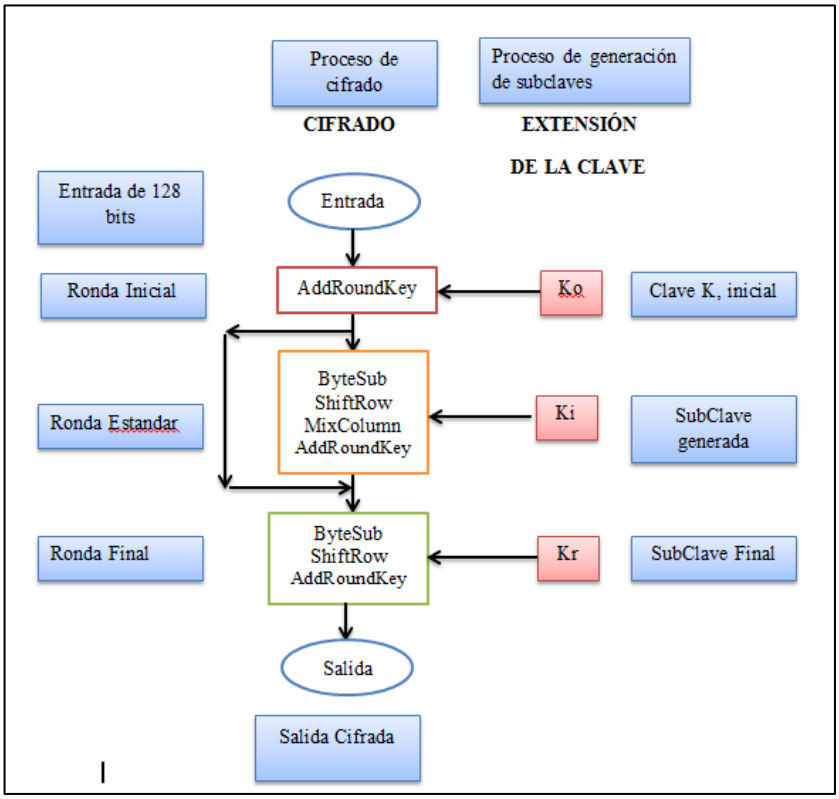
Nk = tamaño de la clave en bits / 32.

A partir de estos cambios la matriz de estado sufre 4 transformaciones por ronda. Las 4 transformaciones que aplica el algoritmo de estado por ronda son: función ByteSub, función ShiftRow, función MixColumn y Función AddRoundKey.

Proceso de Cifrado

Gráficamente la descripción del proceso seria:

Figura 6: Proceso de cifrado del algoritmo de Rijndael



Elaborado por: Jorge Chapaca y David Rojas

El proceso de cifrado consiste en aplicar las cuatro funciones matemáticas. Dichas funciones se realizaran de forma repetitiva para cada ronda.

Características del Algoritmo

- **Flexibilidad**

Rijndael tiene una variedad de bloques y tamaños de claves que pueden ser adaptados en la mayoría de sistemas. Al contrario de lo que sucede con muchos algoritmos, los programadores pueden cambiar la secuencia de un algoritmo. Rijndael para adaptarlo en sistemas más pequeños y agregarlo a plataformas más amplias.

- **Seguridad**

La estructura algebraica que posee Rijndael le permite ser más seguro que un algoritmo promedio. Los usuarios pueden acceder fácilmente a componentes del

programa. Esto le permite a la persona detectar y resolver problemas de seguridad que ocurran en cada evento suscitado. El uso de la revisión de bit y los análisis conductivos del sistema no son necesarios para detectar virus con Rijndael. El algoritmo es especialmente útil defendiendo ataques en energía y tiempo que puedan afectar el sistema.

- **Memoria**

Por ser flexible y defenderse de ataques, Rijndael no requiere mucha memoria para operar. En muchos casos, 128 bits es el máximo requerido. Esto lo hace la elección ideal para software y hardware.

- **La ventaja pública**

Rijndael posee una característica de código abierto. Esta característica permite aumentar la seguridad y la conveniencia para el usuario. En lugar de imprimir documentos y pedir firmas, la criptografía pública permite pedirlas digitalmente. Esto ahorra tiempo, dinero y además guarda de mejor manera los datos. Además, los individuos que eligen documentos firmados digitalmente no deben preocuparse por identificar robo, ya que este algoritmo es muy difícil de infiltrar.

2.6 Seguridad de información en la capa de transporte

La seguridad en la capa de transporte es primordial, ya que es la encargada de garantizar que los destinatarios indiscretos no puedan leer o modificar mensajes dirigidos a otros destinatarios; se preocupa de las personas que intentan acceder a servicios remotos no autorizados. La seguridad también se ocupa del problema de la captura, reproducción de mensajes legítimos y de aquellos que intentan negar que se envíen ciertos mensajes.

Los problemas de seguridad de las redes pueden dividirse en cuatro áreas interrelacionadas las cuales se detallan a continuación:

1. **El secreto y el control de integridad:** Mantiene la información fuera de las manos de los usuarios no autorizados.

2. **La validación de identificación:** Se encarga de determinar con quién se está hablando antes de revelar información delicada o hacer un trato de negocios.
3. **El no repudio:** Se encarga de las firmas: ¿Cómo puede asegurarse de que un mensaje recibido fue realmente enviado, y no algún adversario modificó en el camino su propia cuenta?, Para atacar estos problemas las soluciones deben implementarse en la capa de transporte.
4. **Cifrado Tradicional:** Los mensajes a cifrar son transformados mediante el uso de una clave que tanto el emisor como receptor conocen, por ejemplo si se quiere enviar el texto "HOLA", y la clave es aumentar un número después de cada letra el mensaje cifrado sería H2O5L7A8., (informatica.uv.es, 2010)

Un método alternativo que no necesita modificaciones en los equipos de interconexión es introducir la seguridad en los protocolos de transporte. La solución más usada actualmente es el uso del protocolo SSL o de otros basados en SSL.

Este grupo de protocolos está comprendido en tres fases:

1. **El protocolo de transporte:** secure sockets layer (SSL), desarrollado por netscape communications a principios de los años 90. La primera versión de este protocolo ampliamente difundida e implementada fue la 2.0. Poco después netscape publicó la versión 3.0, con muchos cambios respecto a la anterior, que hoy ya casi no se utiliza.
2. **La especificación:** transport layer security (TLS), elaborada por la IETF (internet engineering task force). La versión 1.0 del protocolo TLS está publicada en el documento RFC 2246. Es prácticamente equivalente a SSL 3.0 con algunas pequeñas diferencias, por lo que en ciertos contextos se considera el TLS 1.0 como si fuera el protocolo "SSL 3.1".
3. **El protocolo:** wireless transport layer security (WTLS), perteneciente a la familia de protocolos WAP (wireless application protocol) para el acceso a la red de dispositivos móviles. La mayoría de los protocolos WAP son adaptaciones de los ya existentes a las características de las comunicaciones inalámbricas, y en particular el WTLS está basado en el TLS 1.0. Las diferencias se centran principalmente en aspectos relativos al uso eficiente del

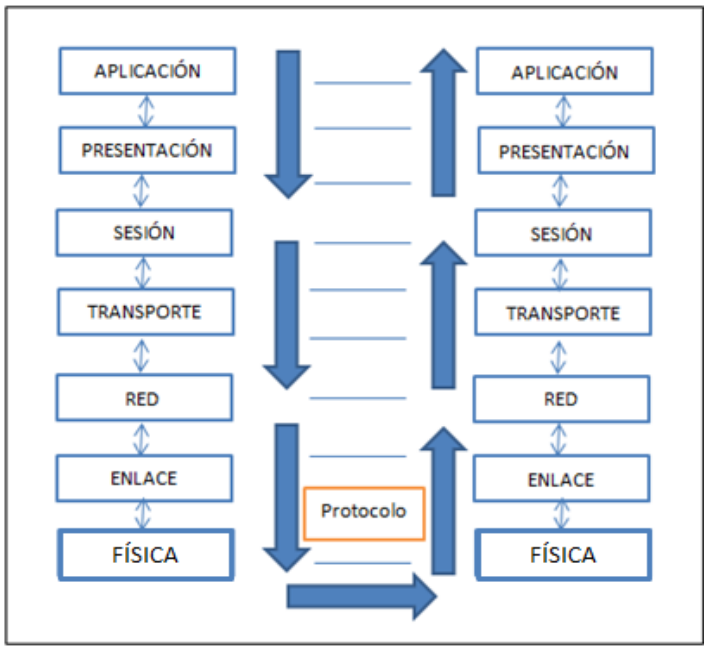
ancho de banda y de la capacidad de cálculo de los dispositivos, que puede ser limitada.

2.7 Arquitectura de comunicación

El proceso de comunicación se produce entre capas equivalentes de dos host distintos, la información va descendiendo por la estructura de capas del host emisor hasta llegar al nivel más bajo, de donde pasa al host receptor y aquí se inicia el viaje ascendente hasta llegar a la capa equivalente en el host de destino.

La capa N de un host emisor se comunica con la capa N de un receptor a través de un protocolo que en mascara el proceso desencadenado en las capas de nivel inferior haciéndolo transparente. La capa 1 opera con transmisiones en el nivel físico, es decir con señales, el resto de las capas opera con comunicaciones.

Figura 7: Arquitectura de Comunicación

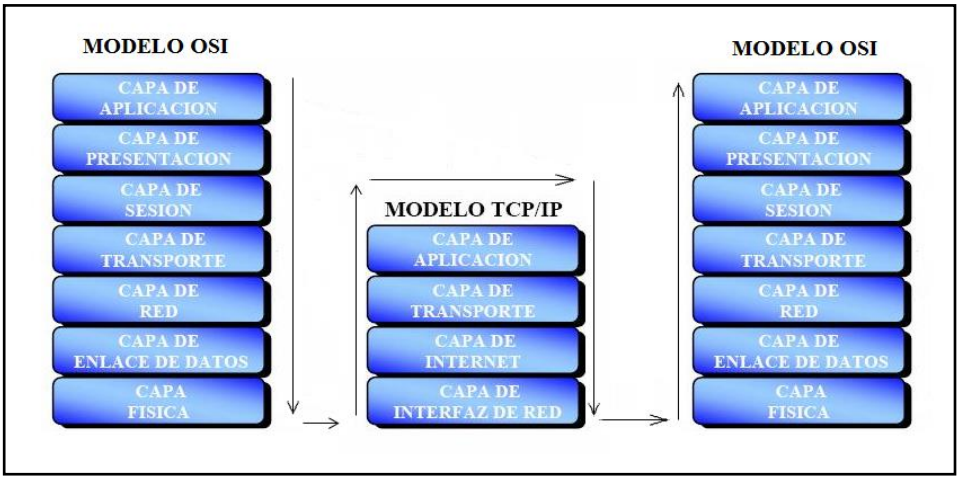


Elaborado por: Jorge Chapaca y David Rojas

- **La comunicación entre las capas**

La comunicación se realiza por interfaces, se da por un sistema de llamadas y respuestas, denominado primitivas. Cada capa ofrece un servicio a través de su interface a la capa superior y está nominado por un SAP que lo identifica únicamente dentro de cada interface.

Figura 8: Arquitectura de Comunicación



Elaborado por: Jorge Chapaca y David Rojas

CAPÍTULO 3
ELABORACIÓN DEL PROTOTIPO DE PROTOCOLO

3.1 Análisis de requerimientos y algoritmos

- **Alcance del prototipo**

Como el prototipo de protocolo no existe, la comunicación se la realizará mediante la utilización de varios routers virtuales (tal como se especifica en las siguientes secciones), los que implementarán la funcionalidad básica del protocolo ETCP.

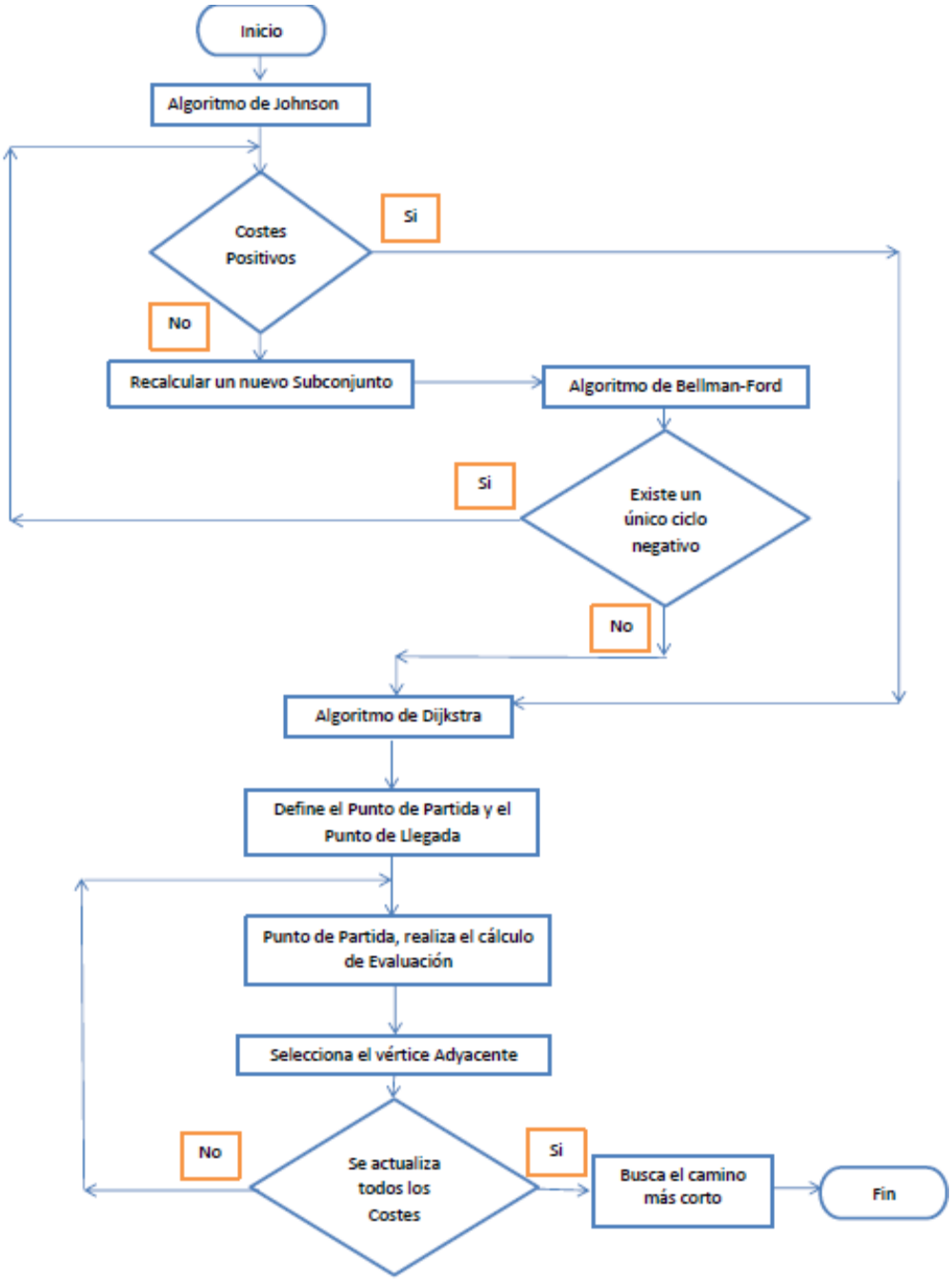
- **Algoritmo de enrutamiento para la funcionalidad del router virtual**

Tomando como referencia las características y comportamiento de los algoritmos de enrutamiento Johnson, Bellman-Ford y Dijkstra detallados en el capítulo 2 sección 2.5, se definió el algoritmo de enrutamiento para la funcionalidad del router virtual.

Se describe a continuación como se encuentra desarrollado el algoritmo tomando como referencia la estructura del algoritmo de Johnson para el arranque del algoritmo de enrutamiento en la reasignación de costes:

- Primero se realiza la evaluación de los costes entre los elementos del grafo, considerando que si todos estos son positivos se usará el procedimiento del algoritmo de Dijkstra para solventar el problema.
- En caso de que existan costes negativos, se recalculará un nuevo subconjunto de costes, para poder usar el paso anterior. Apoyados con el algoritmo de Bellman – Ford, empezando desde donde se quiere encontrar la ruta corta.
- Una vez recalculados los nuevos costes se usa Dijkstra para resolver el problema.

Figura 9: Ciclo del Algoritmo de enrutamiento para la funcionalidad del Router Virtual



Elaborado por: Jorge Chapaca y David Rojas

El resultado con la modificación en los costes entre todos los nodos que intervienen, generan un nuevo grafo de costes positivos; analógicamente al grafo original en el que existía un camino corto para dos puntos, una vez recalculados los costes en el nuevo grafo tendrá el mismo camino corto y viceversa.

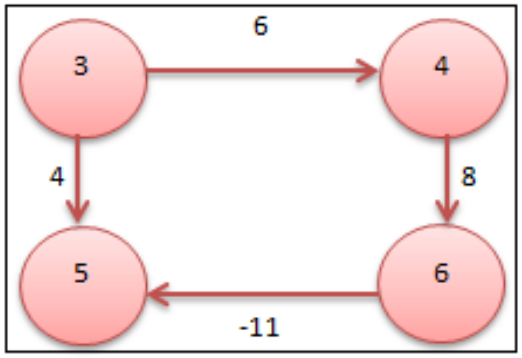
Se puede observar que el algoritmo de enrutamiento definido requiere de la estructura del algoritmo de Bellman Ford y Dijkstra. A continuación se detallará como trabaja el mismo en el momento de encontrar costes negativos:

1. Dado un grafo se detecta si hay un ciclo negativo posible desde el nodo origen hacia el nodo destino.
2. De existir un único ciclo negativo para llegar de un punto a otro, se considera que el grafo no tiene solución.
3. Caso contrario si existen dos o más ciclos negativos se buscará el camino más corto mediante la realización de los siguientes pasos:
    3.1. Se evalúan los costes por cada vértice del grafo.
    3.2. Se relajan los vértices adyacentes actualizando las distancias.
    3.3. Una vez terminadas las iteraciones posibles, se verifica la existencia de un ciclo negativo, para lo cual se vuelve a realizar el paso 3.2 una vez más.
    3.4. Finalmente se define un ciclo negativo para empezar a buscar las rutas más cortas.

Ejemplo 1:

Como se observa en la Figura 10 si se requiere la ruta más corta entre el nodo 3 y 5, el algoritmo de Dijkstra indicaría un coste total de 4 y el camino sería 3 → 5, sin embargo la ruta más corta es por los nodos 3 → 4 → 6 → 5.

Figura 10: Grafo de ejemplo Bellman Ford



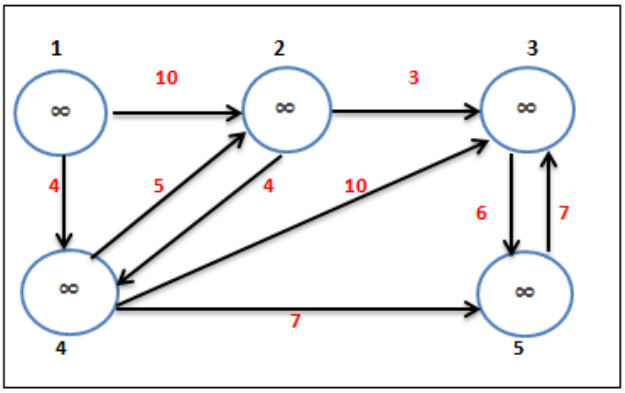
Elaborado por: Jorge Chapaca y David Rojas

Ejemplo 2:

Se requiere ir del nodo 1 hacia el nodo 3.

En el primer paso del algoritmo los costes entre cada arista son infinitos, ya que no se los conoce a primera instancia.

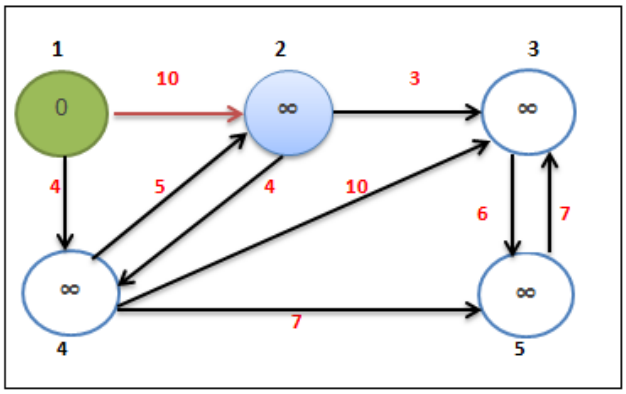
Figura 11: Grafo de ejemplo Dijkstra



Elaborado por: Jorge Chapaca y David Rojas

Primero se evalúa cada vértice adyacente a 1:

Figura 12: Dijkstra paso 1



Elaborado por: Jorge Chapaca y David Rojas

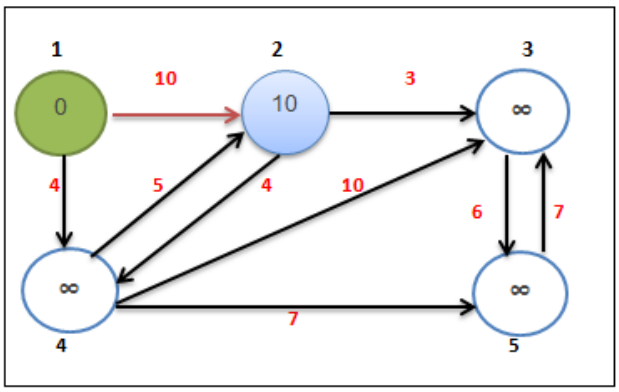
La distancia en el punto 1 es cero, esto debido a que es el vértice origen, por lo tanto la distancia hacia el punto 2 se calcula de la siguiente forma:

$$Si\ D[1] + 10 < D[2] \rightarrow 0 + 10 < infinito.$$

D[n] donde D es la identificación del vértice, y n es el número correspondiente

Como la condición es verdadera entonces el nuevo coste que corresponde del vértice 2 será igual a 10.

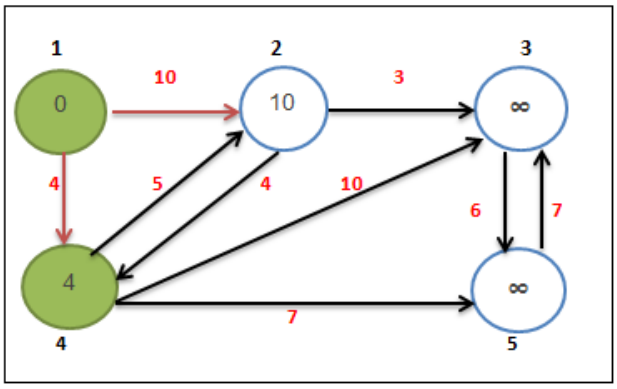
Figura 13: Dijkstra paso 2



Elaborado por: Jorge Chapaca y David Rojas

Se realiza el mismo procedimiento en el vértice 4 que es el siguiente adyacente al vértice 1 obteniendo un coste de 4, tal como se muestra en la Figura 19:

Figura 14: Dijkstra paso 3



Elaborado por: Jorge Chapaca y David Rojas

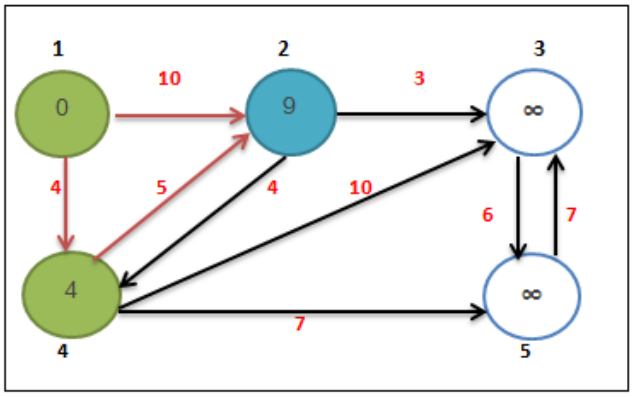
Como el vértice 4 es el de menor coste, entonces se procede a evaluar las adyacencias a este, es decir se evalúan los nodos 2, 3 y 5.

Se evalúa primero el vértice 2, el cual tiene un coste actual de 10, para el cual se realiza el paso de relajación:

$$\text{Si } D[4] + 4 < D[2] \rightarrow 4 + 5 < 10$$

Como la condición se cumple, es necesario actualizar el coste del vértice 2 ya que se encuentra una manera menos costosa (más corta) para llegar a él.

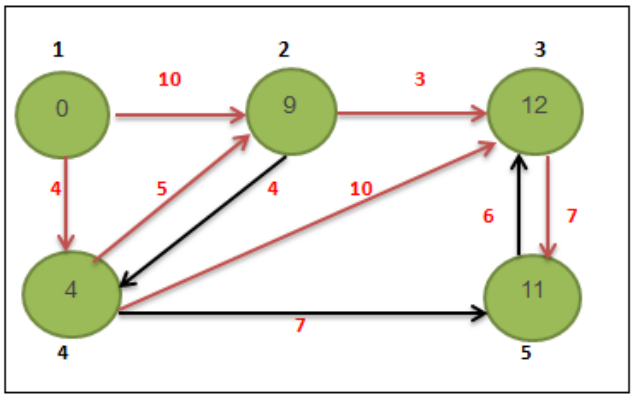
Figura 15: Dijkstra paso 3



Elaborado por: Jorge Chapaca y David Rojas

Del mismo modo se procede a evaluar todos los vértices del grafo, teniendo como resultado:

Figura 16: Dijkstra paso 4

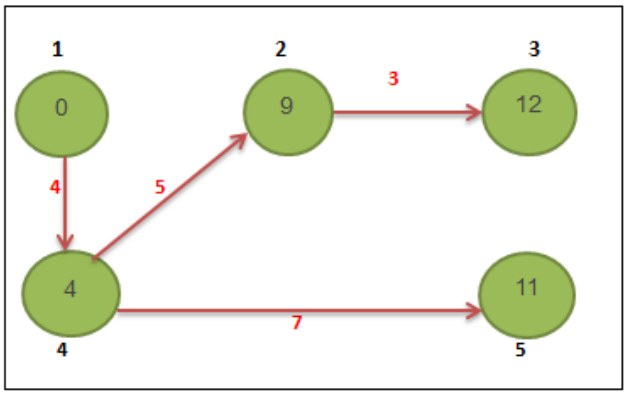


Elaborado por: Jorge Chapaca y David Rojas

Finalmente se evalúan las rutas más cortas explicado en el siguiente grafo:

- El camino más corto y optimo seria por el nodo 1, nodo 4, nodo 2, y finalmente el nodo 3, la explicación es porque esos son los menores costes que se pueden dar en el grafo.

Figura 17: Dijkstra paso final



Elaborado por: Jorge Chapaca y David Rojas

- **Implementación del algoritmo de enrutamiento**

En la implementación del algoritmo de enrutamiento para la funcionalidad del router virtual, se adoptan los pasos de Johnson, Bellman-Ford conjuntamente con los de Dijkstra.

El algoritmo de enrutamiento para el router virtual se encuentra implementado en la clase RouterServer.cs, sus métodos y funciones están detalladas en el **Anexo 2**.

Existen algunas líneas de código importantes en la implementación del algoritmo las cuales se detallan a continuación:

- Primero se verifica que no existan dos nodos iguales a través de la siguiente línea de código:

```
if (Paths.Any(p => p.Source.Equals(p.Destination)))
throw new ArgumentException("No path can have the same source and
destination");
```

- Mediante las siguientes sentencias se crea una lista con los posibles caminos.

```
Paths.SelectMany(p => new T[] { p.Source, p.Destination })
.Distinct()
.ToList()
.ForEach(s => ShortestPaths.Set(s, Int32.MaxValue, null));
```

- Mientras no se hayan procesado todos los posibles caminos, se sigue interactuando los nodos posibles.

```
    while (LocationsProcessed.Count < LocationCount)
    {
    T _locationToProcess = default(T);
    foreach (T _location in ShortestPaths.OrderBy(p =>
p.Value.Key).Select(p => p.Key).ToList())
    {if (!LocationsProcessed.Contains(_location))
    {if (ShortestPaths[_location].Key == Int32.MaxValue)
    return ShortestPaths.ToDictionary(k => k.Key, v => v.Value.Value);
    _locationToProcess = _location;
    break;
    }
    } // foreach
```

- Una vez encontrada la ruta se calculan los costes.

```
    var _selectedPaths = Paths.Where(p =>
p.Source.Equals(_locationToProcess));
    foreach (Path<T> path in _selectedPaths)
    {
    if (ShortestPaths[path.Destination].Key > path.Cost +
ShortestPaths[path.Source].Key)
    {
    ShortestPaths.Set(
    path.Destination,
    path.Cost + ShortestPaths[path.Source].Key,
    ShortestPaths[path.Source].Value.Union(new Path<T>[] { path
}).ToArray());
    }
    }

    LocationsProcessed.Add(_locationToProcess);

    }

    return ShortestPaths.ToDictionary(k => k.Key, v => v.Value.Value);
    }
    }
```

➢ Interfaz del router virtual

El router virtual presenta una interfaz de comunicación para los usuarios y para las aplicaciones que lo implementen, teniendo en ambos casos como punto final, la traducción a la interfaz IP, siendo esta la de más bajo nivel para la implementación.

La interfaz entre procesos de aplicación y el protocolo se representan como un conjunto de llamadas similar a las que un sistema operativo proporciona, por ejemplo

habrá llamadas para abrir, cerrar conexiones y enviar datos. Adicional se deja abierta la posibilidad de crear nuevas llamadas, según sea necesidad.

- **Proceso de conexiones del router virtual**

Para permitir muchas conexiones y procesos dentro de un mismo host se usan de manera simultánea la comunicación, el router virtual proporciona una serie de direcciones y puertos para poder conformar los sockets de conexión, de esta manera un par (dirección IP - puerto) identifica una única forma de conexión.

La asignación de direcciones y puertos en cada proceso se gestiona de manera independiente en cada terminal. Sin embargo para procesos frecuentes es necesario establecer un par único y fijo para darlos a conocer de forma pública.

3.2 Definición del Prototipo de Protocolo

ETCP es un prototipo de protocolo de transporte ha sido elaborado para la transmisión de datos comunicados entre redes de distintos lugares cubriendo las necesidades de transferencia de datos seguros, confiables y de alta disponibilidad.

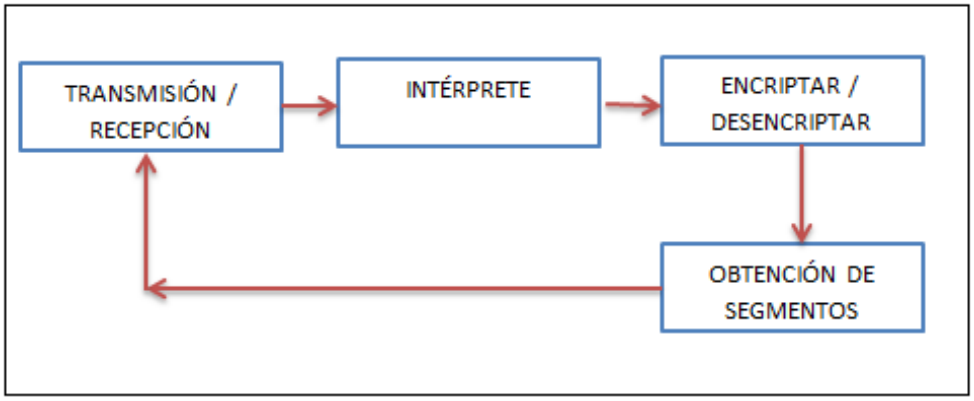
- **Funciones del prototipo de protocolo ETCP**

  - Permite la transmisión y recepción de segmentos
  - Implementa la encriptación necesaria para poder realizar el envío seguro de los segmentos
  - Capacidad para transmitir segmentos de manera continua, evitando que se tenga que realizar una nueva solicitud para su retransmisión

- **Diseño de la arquitectura del prototipo de protocolo**

Para el flujo de transmisión y recepción de segmentos se tiene los siguientes procesos:

Figura 18: Procesos del protocolo



Elaborado por: Jorge Chapaca y David Rojas

- **Transmisión – Recepción**

El objetivo de este proceso es el de transmitir y receptar las unidades de información llamadas "segmentos", que sean compatibles con el nuevo prototipo de protocolo ya que serán filtrados y posteriormente enviados a la siguiente capa intérprete. La transmisión y recepción de segmentos se lo puede apreciar en el **Anexo 2** en el método start y **Anexo 3** en el método sendMessage.

- **Intérprete**

Cada protocolo tiene su forma de interpretar los segmentos para poder transmitirlos y recibirlos, la funcionalidad de este proceso es de realizar una traducción e interpretación de los segmentos que deben ser transmitidos o recibidos por ETCP de aquellos protocolos que no tengan relación con él propuesto como de sí mismo.

El funcionamiento del intérprete se lo puede apreciar en el **Anexo 5.**

- **Encriptación – Desencriptación**

La funcionalidad de éste proceso es la ejecución de los algoritmos de encriptación y desencriptación para la transmisión de los segmentos que viajan por la red, se traducirá los datos de una forma en la que se pueda entender el protocolo usado, y cuando se requiere de una mayor seguridad se puede aplicar un esquema de cifrado.

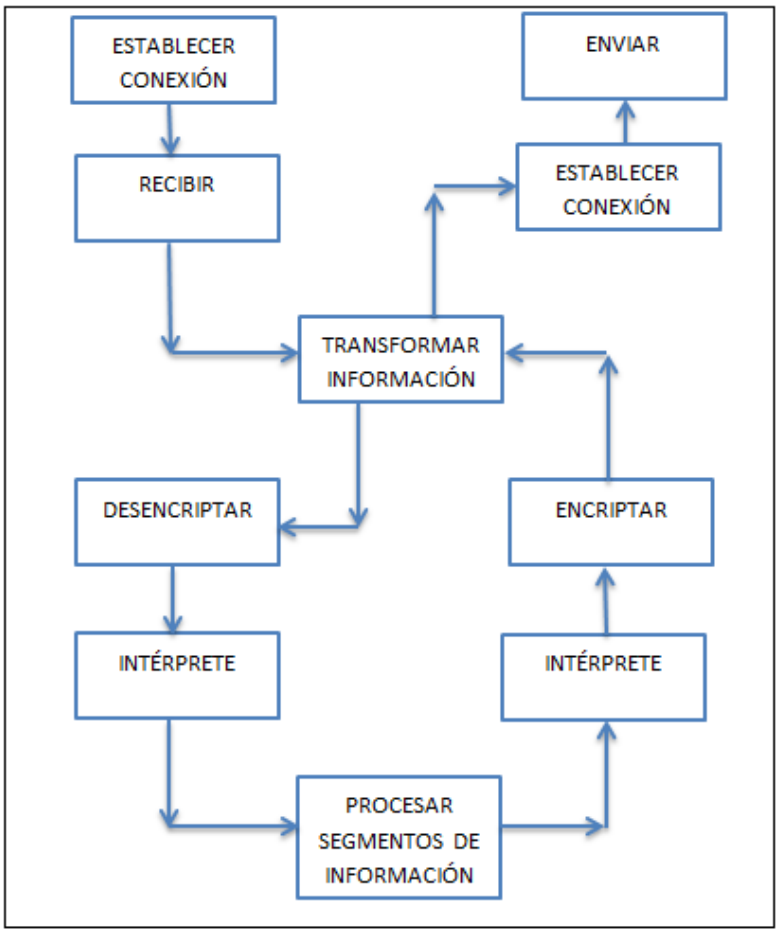
Se debe recordar que todo lo que hace el lado de la transmisión tiene su opuesto correspondiente en la recepción de segmentos, tal como se lo puede apreciar en el **Anexo 4.**

- **Obtención de segmentos**

Una vez realizado la encriptación y desencriptacion de los segmentos, se extrae la información previamente procesada en la "data" que se envió desde un extremo de la comunicación (recepción), o se realizará un ensamblado de la data en el mensaje a enviar (transmisión). El funcionamiento de la obtención de segmentos se puede apreciar en el **Anexo 2.**

Del diagrama propuesto se tiene que el sistema obedece a un flujo de trabajo por el cual se debe atravesar para poder procesar y usar la información transmitida, tal como se puede apreciar en la siguiente figura:

Figura 19: Flujo del prototipo



Elaborado por: Jorge Chapaca y David Rojas

De acuerdo al flujo del prototipo se establece las funciones para el envío y recepción de datos

- **Funcionamiento del envío**
  - Procesar la información del servidor
  - Se interpreta la información a enviar
  - Se encripta la información
  - Se realiza la segmentación
  - Se envía la información por el canal de comunicación que fue previamente abierto

- **Funcionamiento de la recepción**
  - Se establece la conexión con el nodo a comunicarse
  - Se transforma la data recibida, es decir se toman los bytes entrantes y se transforma a una cadena de texto
  - Se desencripta el contenido del mensaje
  - Se traduce a modo que ETCP puede entender; para esto se realiza la interpretación del mensaje completo
  - Finalmente se procesa la información dentro del servidor

3.2.1 Especificación del protocolo

A continuación se describe la especificación que debe mantener el protocolo de comunicación:

- **Operación**

El propósito del protocolo es mantener una comunicación fiable y continua entre procesos. Para proporcionar el servicio el sistema de comunicación se requiere de mecanismos relacionados con las siguientes áreas:

- Transferencia de segmentos
- Control de flujo
- Fiabilidad
- Confiabilidad
- Conexiones
- Alta Disponibilidad
- Encriptación

- **Transferencia de segmentos**

ETCP tiene la capacidad de transmitir un flujo continuo de datos en cada sentido, empaquetando los octetos para posteriormente segmentarlo para lograr una perfecta transmisión a través del sistema de internet o la intranet.

El código permite la transferencia de segmentos se describe a continuación:

- Obtener el contenido del mensaje

```
messageData.MessageContent = messageData.MessageContent;
```

- Obtener la instancia del stream de red

```
NetworkStream serverStream = tcpClient.GetStream();
```

- Asignar timeouts para la lectura y escritura, esto para dar un cierto tiempo para que no se cuelgue el programa

```
serverStream.ReadTimeout = ReadTimeout;
serverStream.WriteTimeout = WriteTimeout;
```

- Transformar a un arreglo de bytes, el contenido del mensaje, y finalmente se lo envía a la red

```
byte[] outStream =
System.Text.Encoding.ASCII.GetBytes(messageData.GetString())
;
serverStream.Write(outStream, 0, outStream.Length);
serverStream.Flush();
tcpClient.Close();
```

- **Control de flujo**

ETCP proporciona al receptor el medio para controlar la cantidad de segmentos enviados, esto se obtiene mediante la respuesta inmediata cada vez que se haga la recepción de la data, indicando al emisor cual es el siguiente segmento a enviar.

- **Fiabilidad**

ETCP es capaz de recuperar los segmentos que se corrompan o se pierdan durante el proceso de transmisión de datos. Aún si los segmentos no llegasen en orden el sistema será capaz de reconstruir el mensaje original, esto se consigue asignando un identificador único de secuencia a cada segmento enviado. Si no se recibe uno de los segmentos en un tiempo determinado, entonces ETCP vuelve a realizar la petición del segmento faltante.

- **Confiabilidad**

La capa de transporte puede implementar un método para asegurar la entrega confiable de los datos. En términos de redes, confiabilidad significa asegurar que cada sección de segmentos que se envié llegue al destino. En la capa de transporte, las tres operaciones básicas de confiabilidad son:

- Rastreo de segmentos transmitidos
- Acuse de recibo de segmentos recibido
- Retransmisión de cualquier segmento sin acuse de recibo

Confiabilidad requiere que todos los segmentos transmitidos lleguen al destino en su condición original, de manera que los mismos sean útiles. Todos los segmentos perdidos pueden corromper una comunicación y dejarla incompleta o ilegible. Por lo tanto, se crea protocolos de capa de transporte que implemente la confiabilidad como lo realiza ETCP.

- **Conexiones**

La fiabilidad y el control de flujo exigen que se inicialicen todos los procesos de ETCP y de esta manera se mantenga actualizada la información del estado de cada proceso. Cuando dos procesos deseen comunicarse deben establecer primero una conexión (inicializar los módulos en cada extremo). Cuando la comunicación se haya completado, la conexión se cerrará para liberar los recursos utilizados y que nuevas conexiones puedan usarlos.

- **Alta disponibilidad**

La alta disponibilidad es la característica que tiene un sistema para protegerse de interrupciones o caídas, de forma automática y en un corto plazo de tiempo. Los

sistemas de alta disponibilidad se diseñan para eliminar o tolerar los puntos de fallo que posiblemente puedan ocurrir al momento de la transmisión. Por esa razón se emplea principalmente la redundancia interna de componentes (red, almacenamiento, fuentes de alimentación, etc.) así como de los elementos de infraestructura (sistema eléctrico, electrónica de red, etc.).

- **Encriptación**

ETCP pueden especificar un nivel de seguridad y establecer la prioridad de la comunicación para lo cual se emplearán valores por defecto en caso de no indicar estos parámetros. Para el efecto se configuran métodos que implementan el algoritmo Rijnadel de encriptación.

La implementación se realiza mediante la clase EncodeManager.cs, para ver la funcionalidad, observar el **Anexo 4.**

3.2.2 Especificación funcional de ETCP

- **Formato del mensaje**

Cada segmento del mensaje se enviarán a manera de texto plano a través de la red, así la cabecera está formada de la siguiente manera:

- 18 bits para la dirección IP y puerto de origen
- 18 bits para dirección IP y puerto de destino
- 1 bits para indicar si se trata de un segmento entero o es parte de un conjunto de segmento, 0 significa que es un mensaje completo, y 1 que aún necesita más segmento por llegar
- 8 bits para el secuencial de datos, el mismo que estará presente en caso de que el elemento anterior este activado
- 8 bits para el secuencial del siguiente segmento que deberá recibir, el mismo que estará presente en caso de que el elemento anterior este activado
- 3 bits de ejecución, indican el método de respuesta por parte de quien recibe el mensaje

A continuación el formato del mensaje, los dos primeros significan la acción del emisor, el tercer bit la acción que debe realizar el receptor (C → Enviar confirmación de recepción, F → No necesita confirmación de recepción).

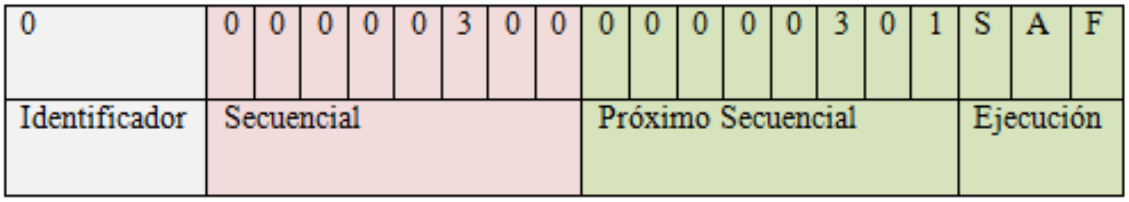
Tabla 4: Cuadro de equivalencias de bits de ejecución

Valor 2 primeros bits	Significado	Tercer bit posible
EA	Último segmento en enviarse.	C, F
RS	Reiniciar el envío.	C, F
RC	Reiniciar conexión.	C, F
SS	Faltan más segmentos, envío síncrono.	C, F
	No importa el orden de recepción.	
SA	Faltan más segmento, envío asíncrono.	C, F
	Orden de recepción es importante.	
PC	Petición de conexión.	C
EC	Petición de finalización de conexión.	F

Elaborado por: Jorge Chapaca y David Rojas

En total serán 56 bits para la cabecera. El resto estarán empaquetadas en un nivel básico que se explicará en el siguiente punto.

Figura 20: Ejemplo



Elaborado por: Jorge Chapaca y David Rojas

- **Establecimiento de conexión**

El procedimiento se origina a partir de dos puntos que no esté previamente conectados, para asegurar que dos procesos no tratan de abrir una conexión

simultánea y bloqueen la conexión, se tiene un canal especial para la recepción de peticiones de conexión.

Para establecer la conexión se definen los siguientes pasos:

- Envío de petición
- Espera por respuesta
- Recepción o timeout
- En caso de recepción se establece el canal de comunicación, caso contrario se realiza otra petición o se finaliza el intento

En caso de que se trate de arrancar una conexión de manera simultánea entre los dos puntos, primero los host deben ver si no existe ese intento en sus registros de conexión, caso contrario no se le hará caso al intento, y seguirá establecida la conexión.

- **Cierre de conexión**

La función end_connection, la cual enviará un mensaje de petición de cierre de conexión desde el emisor al receptor, de esta manera se cortará la comunicación y liberaran los recursos del sistema.

En el caso de que el receptor y emisor emitan al mismo tiempo mensajes de desconexión, se tomará la misma consideración que en la conexión. Primero se verifica que no exista una solicitud de desconexión pendiente, en caso de existir el sistema tomará una de las peticiones y rechazará la otra.

- **Seguridad**

La intención es que la conexión sólo se permita entre puertos que operen exactamente con los mismos valores de seguridad que fueron optados anteriormente, un intento de conexión con valores de seguridad erróneo debe ser rechazado enviando una petición de cierre o de reseteo de la misma.

- **Comunicación de datos**

Una vez establecida la conexión los datos se transmiten mediante el intercambio de segmentos. Dado que los segmentos pueden perderse debido a errores (fallo en la

suma de comprobación) o congestión de la red, el protocolo utiliza la retransmisión (tras un tiempo de espera) para asegurar la entrega de cada segmento. Pueden llegar segmentos duplicados debido a la red o a la retransmisión de datos. En este caso el protocolo realiza ciertas comprobaciones sobre los números de secuencia y de recibo en los segmentos para verificar su admisibilidad.

El emisor de los datos mantiene un registro del próximo número de secuencia a usar. El receptor de los datos mantiene un registro del siguiente número de secuencia esperado. Si el flujo de datos queda momentáneamente inactivo y de todos los datos enviados se tiene la respuesta de recepción, al reiniciar existe la posibilidad de duplicar data, por lo que el receptor dará la señal de restablecer la conexión pidiendo segmentos que falten para completar la transferencia. Se puede revisar el A**nexo 2 y Anexo 3.**

- **Tiempo de espera para retransmisión**

Debido a la gran variedad de redes que componen el internet y los diferentes protocolos que se aplican para la comunicación es necesario que exista un parámetro de espera para retransmitir, el mismo que se deberá determinar dinámicamente.

Para realizar este cálculo se toma en cuenta el tiempo de respuesta del primer segmento enviado, de esta manera se tiene un tiempo de referencia, al que se incrementará un tiempo configurable en el protocolo para tenerlo como tiempo de espera, en resumen se realizará la suma entre el promedio del tiempo de transmisión más el parámetro configurable.

```
Tr = PROM(T1 + T2 + …. Tn) + retransmition_time
```

La implementación de los tiempos de espera está en el **Anexo 3.**

- **Ordenes de usuario (usuario / protocolo)**

Las siguientes secciones caracterizan de forma funcional una interfaz usuario/protocolo.

La notación utilizada es similar a la mayoría de las llamadas para lograr proceder o funcionar en los lenguajes de alto nivel.

Las órdenes de usuario descritas más abajo especifican las funciones básicas que debe ejecutar el protocolo para soportar la comunicación entre procesos.

Las implementaciones individuales deben definir su propio formato exacto, y pueden proporcionar combinaciones o subconjuntos de las funciones básicas en llamadas simples.

En particular, algunas implementaciones pueden querer realizar la llamada de apertura open de la conexión automáticamente en el primer comando enviado por el usuario para una conexión dada.

Al proporcionar recursos para la comunicación entre procesos, el Protocolo no debe limitarse a aceptar órdenes, sino que también debe devolver información al proceso que sirve. Esta información consistirá en:

- Información general acerca de una conexión (Parámetros de enlace, tiempo de conexión, etc.).
- Respuestas a órdenes de usuario específicas indicando éxito o varios tipos de errores.

Las funciones que dispone el protocolo para la comunicación, con la aclaración que los elementos entre corchetes [] son opcionales, en caso de no ser enviados se tomarán los parámetros por defecto.

La forma de enviar los parámetros es la bandera que identifica al parámetro seguida del parámetro, ejemplo -p 5000, donde –p es el identificador del parámetro y 500 es el valor.

- **Open**

Parámetros:

Tabla 5: Parámetros de los comandos open

Parámetro	Descripción	Ejemplo	Opcional
-p	Puerto local.	-p 5000	NO
-rc	Conexión remota. Compuesta por IP y puerto remoto.	-rc 192.0.0.1:3000	NO
-wt	Tiempo de espera, en milisegundos. Valor por defecto 1000.	-wt 200	SI
-pr	Prioridad del enlace. Por defecto 1.	-pr 10	SI
-sp	Protocolo de seguridad. Por defecto seguridad baja (ces). Opciones: Des Data EncryptionStandar Ces Cifrado César Rij Rijndel	-spdes	SI
-n	Nombre de la conexión. Por defecto se llamará Conection_#, Siendo # el número de conexión.	-n ConexionPrueba	SI

Elaborado por: Jorge Chapaca y David Rojas

Una orden open activa proceso de conexión tanto en el remitente como en el receptor, quedando registrada la conexión y reservados los puertos implicados.

Si la conexión permanece inactiva por un tiempo determinado (2 minutos por defecto), esta será cerrada para liberar los recursos no utilizados.

El nombre que se asigna a la conexión puede ser usado luego a manera de atajo para no reescribir de nuevo todos los parámetros que dieron inicio a dicha conexión, ya que el sistema los almacenará para usos posteriores.

Tabla 6: Consideraciones de los comandos open

Estado del receptor	Procedimiento
Cerrado	Se devuelve error en caso de que no se haya solicitado usar la fuerza para abrir la conexión
Escuchando	Simplemente se establece la conexión y pasa de estado escucha a estado abierto.

Elaborado por: Jorge Chapaca y David Rojas

En el caso de que se solicite abrir una conexión que ya está abierta se devolverá un mensaje de error que la conexión ya existe.

- **Send**

Tabla 7: Parámetros del comando Send

Parámetro	Descripción	Ejemplo	Opcional
-n	Nombre de la conexión.	-n Conexión Prueba	NO
-d	Data a enviar. Usado para envió de mensajes planos.	-d “Hola”	NO
-df	Archivo a enviar. Path del archivo a enviar.	-df “c:\archivo.doc”	NO
-t	Tiempo de espera en milisegundos. Cuanto puede demorarse en recibir la confirmación del envío, defecto 200.	-t 1000	SI
-ca	Algoritmo de compresión. Por defecto sin comprimir. Opciones: Zip Rar	-cazip	SI
-fc	Forzar conexión. Por defecto no fuerza a la conexión. Opciones: Y si N no	-fc y	SI

Elaborado por: Jorge Chapaca y David Rojas

Esta orden se encarga de enviar datos de un terminal a otro, para lo cual las banderas –d y –df son necesarias, una de las dos, no ambas al mismo tiempo.

En caso de que la conexión no haya sido abierta aún, el protocolo envía un mensaje de error al procesar la orden, ya que primero se debe abrir la conexión para transmitir datos; a menos que envíe la bandera –fc la cual si no existe conexión forzará para que se logre abrir la transmisión de datos.

En la implementación más simple, la llamada send no devolverá el control al usuario hasta que se complete la transmisión o se supere el tiempo de espera. Sin embargo, este método simple está sujeto a puntos muertos ('deadlocks') (por ejemplo, ambos lados de la conexión podrían intentar realizar operaciones send antes de hacer ninguna operación de recibir) y ofrece un rendimiento pobre, por lo cual no se recomienda. Una implementación más sofisticada devolverá el control inmediatamente para permitir al proceso ejecutarse concurrentemente con la E/S de red y, además, permitir que múltiples operaciones send tengan lugar a la vez. Las operaciones send múltiples son atendidas según van llegando, de forma que el protocolo pondrá en cola aquellas a las que no puede atender inmediatamente.

- **Receive**

Tabla 8: Parámetros del comando receive

Parámetro	Descripción	Ejemplo	Opcional
-n	Nombre de la conexión.	-n Conexión Prueba	NO

Elaborado por: **Jorge Chapaca y David Rojas**

Ésta orden inicializa un búfer de recepción asociado con la conexión especificada. Si esta orden no viene precedida de una llamada OPEN o el proceso solicitante no está autorizado a usar esta conexión, se devuelve un error.

En la implementación más simple, el control no volverá al programa solicitante hasta que el búfer se llene u ocurra algún error, pero este esquema tiene un alto riesgo de puntos muertos. Una implementación más sofisticada permitiría que varios receive se ejecutaran a la vez. Estos se irían completando según van llegando los segmentos.

Esta estrategia permite el rendimiento a costa de un esquema más elaborado (posiblemente asíncrono) para notificar al programa solicitante que se ha detectado una llamada de entrega inmediata o se ha llenado un búfer.

En caso de que el paquete recibido sea solo una parte de un paquete grande, los datos que vayan llegando serán almacenados en un repositorio para su posterior ensamblaje y arma de del paquete grande.

- **Close**

Tabla 9: Parámetros del comando close

Parámetro	Descripción	Ejemplo	Opcional
-n	Nombre de la conexión.	-n Conexión Prueba	NO

Elaborado por: Jorge Chapaca y David Rojas

Esta orden cierra la conexión especificada. Si la conexión no está abierta o el proceso solicitante no está autorizado a usar dicha conexión, se devuelve un error. El cierre de conexiones está pensado para que sea una operación limpia en el sentido de que las llamadas send pendientes serán transmitidas (y retransmitidas), en la medida en que el control de flujo lo permita, hasta que todas hayan sido servidas. Por tanto, es aceptable enviar varias órdenes send seguidas de una llamada close, y suponer que todos los datos serán enviados a su destino. Debe quedar claro que los usuarios continúan recibiendo por conexiones que se está cerrando, ya que el otro lado de la conexión transmite el resto de sus datos.

Por tanto, cerrar una conexión significa "no tengo nada más que enviar" pero no "no recibiré nada más". Puede suceder (si el protocolo del nivel de usuario no está bien ideado) que el lado de la conexión que cierra no sea capaz de deshacerse de todos sus datos antes de que se cumpla el tiempo de espera. En este caso, la llamada close se convierte en una llamada abort y el protocolo que cierra la conexión deja de seguir.

El usuario puede cerrar la conexión en cualquier momento bajo su propia iniciativa o en respuesta a varios avisos del protocolo (ejemplo: ejecutado un cierre remoto, excedido el tiempo de espera de transmisión, destino inaccesible). Dado que cerrar

una conexión requiere la comunicación con el protocolo remoto, las conexiones permanecen en el estado de cierre durante un corto lapso de tiempo. Los intentos de reabrir la conexión antes de que el protocolo responda a la orden close darán como resultado respuestas de error.

- **Status**

Tabla 10: Parámetros del comando status

Parámetro	Descripción	Ejemplo	Opcional
-n	Nombre de la conexión.	-n Conexión Prueba	NO

Elaborado por: Jorge Chapaca y David Rojas

Devuelve el estado de la conexión.

- **Abort**

Tabla 11: Parámetros del comando abort

Parámetro	Descripción	Ejemplo	Opcional
-n	Nombre de la conexión.	-n Conexión Prueba	NO

Elaborado por: Jorge Chapaca y David Rojas

Abortará la ejecución de las siguientes órdenes:

- Send
- Close

CAPÍTULO 4
ROUTER VIRTUAL CON FUNCIONALIDAD BÁSICA

4.1 Análisis de requerimientos

El prototipo se encuentra desarrollado bajo tecnología microsoft utilizando Visual Studio .NET 2010 y C#, de la definición del protocolo, así como de su especificación, que implementará la funcionalidad básica en el router con el fin de cumplir enrutamiento, confiabilidad, disponibilidad y seguridad.

El router virtual desarrollado en su totalidad cuenta con la capacidad de informar en tiempo real las diferentes situaciones que ocurran durante la ejecución de ETCP.

Para esto se creó un sistema de manejo de bitácora, el mismo que registrará los eventos que sucedan durante el tiempo de ejecución del simulador.

4.2 Elaboración del router virtual

Para la elaboración del router virtual se usó visual studio 2010.

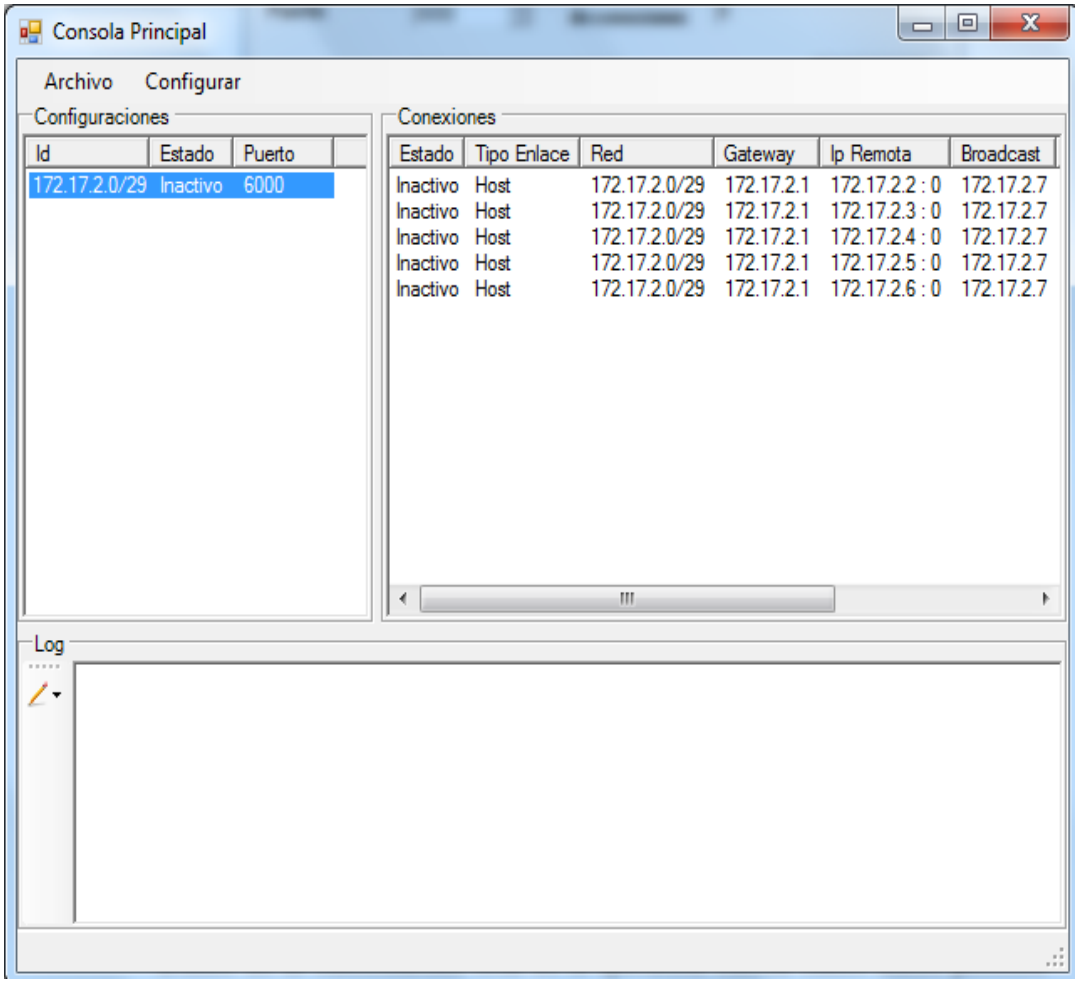
Interfaz que proporciona un entorno gráfico intuitivo para el desarrollo y programación de formularios.

Para las pruebas se realizó dos interfaces principales, una para el router y otra para los host, como se trata de un prototipo base, las interfaces permiten el envío de mensajes entre los distintos puntos de las redes que se conformen.

A continuación se detallan las dos interfaces principales del prototipo:

Interfaz del router:

Figura 21: Interfaz del Router



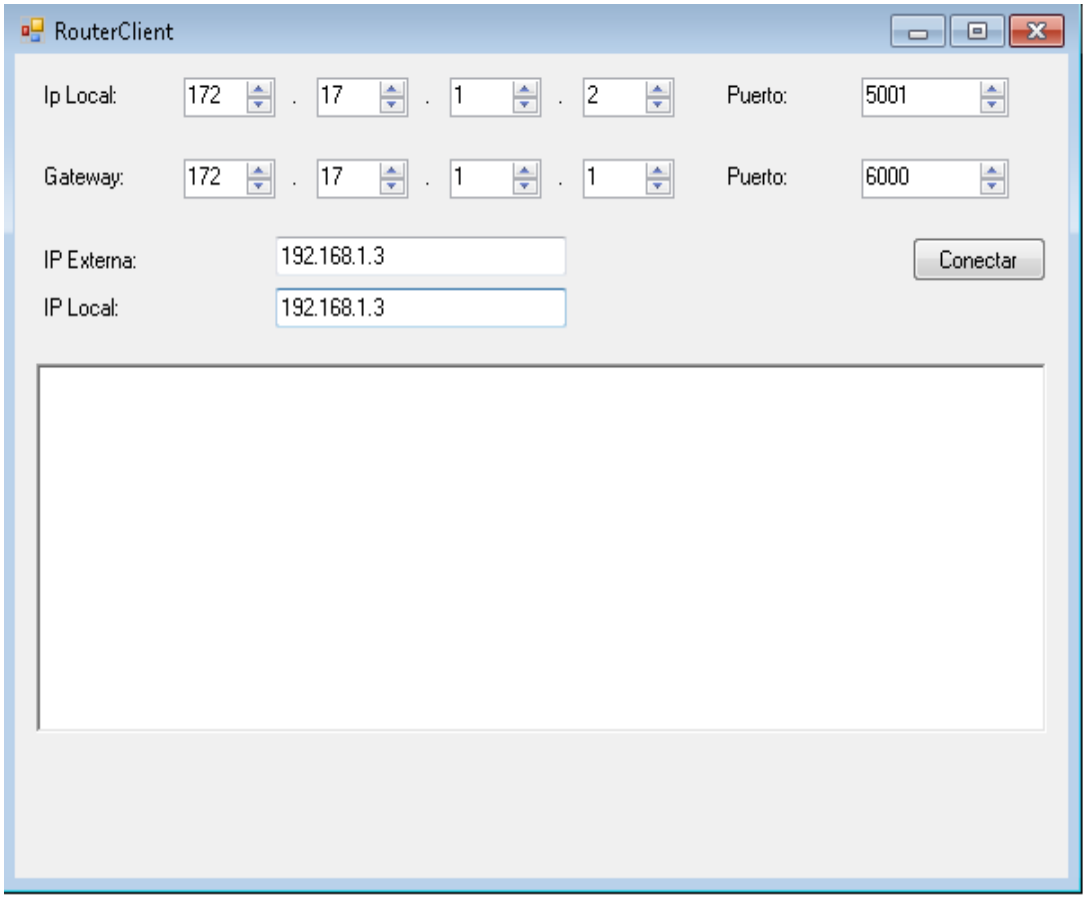
Elaborado por: Jorge Chapaca y David Rojas

El cuadro inferior servirá para desplegar la bitácora de ejecución del software.

El panel superior izquierdo para mostrar las redes configuradas o las que se encuentran ya conectadas, mientras que la parte superior derecha muestra los puntos de acceso entre routers y terminales.

Interfaz del cliente (host):

Figura 22: Interfaz de host



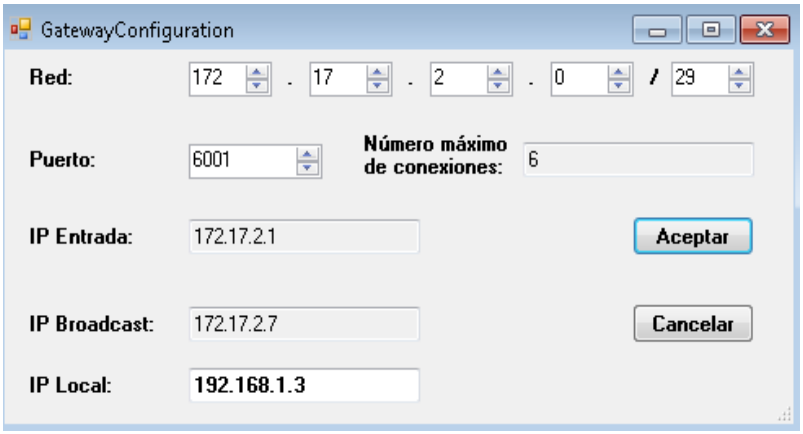
Elaborado por: Jorge Chapaca y David Rojas

De igual manera que la interfaz del router, la parte inferior servirá para el despliegue de los datos de la bitácora generada durante la ejecución. La parte superior sirve para ingresar los parámetros de conexión hacia el router.

Además de las dos interfaces, cuenta con sub interfaces que ayudan a la configuración y conexión del router.

Sub Interfaz de creación de red:

Figura 23: Sub Interfaz de creación de red



Elaborado por: Jorge Chapaca y David Rojas

Contiene los parámetros para configurar una red, siendo campos obligatorios la IP de red, la máscara y el puerto en el que se va a levantar para la ejecución. Al ser un protocolo que no es parte de ningún sistema operativo, se genera la necesidad de crear dichos simuladores, por eso como detalle adicional siempre se pide el puerto en el cual se va a escuchar a las conexiones.

El prototipo de router está en la capacidad de crear automáticamente las conexiones posibles, como se muestra en el siguiente ejemplo. Se levanta una red de hosts, para esto se hace que el router levante una red, para éste caso se lo va a hacer con la siguiente dirección de red: 172.17.2.0/29 como identificador de red, 172.17.2.1/29 como IP de gateway, luego el intervalo de redes desde 172.17.2.2/29 a 172.17.2.6/29 como IP de host disponibles y finalmente 172.17.2.7/29 como IP de broadcast.

Para este caso se realiza los cálculos de red respectivos se obtienen las direcciones 192.168.69.0/26 como identificador de red, 192.168.69.1/26 como IP de gateway, luego el intervalo de redes desde 192.168.69.2/26 a 192.168.69.62/26 como IP de host disponibles y finalmente 192.168.69.63/26 como IP de broadcast.

Para una mejor comprensión del protocolo se procede a la construcción de interfaces gráficas de usuario, cuya finalidad es visualizar e interactuar con el usuario para que pueda comprender el propósito del protocolo. La elaboración de las interfaces se

trabajó bajo el criterio de necesidad, es decir, se presentan datos que se los toma como importantes o que son necesarios para comprender la funcionalidad del sistema.

4.3 Integración e implementación de la especificación del protocolo

Para la programación de la implementación se seleccionaron las siguientes funcionalidades:

- **Interfaz de router**

Interfaz gráfica correspondiente a la implementación del servidor principal, aquí se usan todos los métodos del **Anexo 2** que corresponden al desarrollo del protocolo ETCP.

- **Interfaz de cliente**

Interfaz gráfica correspondiente al cliente de comunicaciones, cuyo uso se lo puede apreciar en el **Anexo 1** correspondiente al manual de usuario.

- **Envío de mensajes cliente – router – cliente**

Corresponde a la implementación para envío de mensajes hacia los diferentes destinos. Para esto se implementaron métodos y funciones que se pueden apreciar en el **Anexo 3** que corresponde a la clase encargada de realizar el proceso de envío.

- **Parsing de mensajes, con método de posicionamiento**

Implementa un algoritmo básico de parseo posicional, el mismo que se puede apreciar en el **Anexo 5**, en cuyo contenido se encuentran las funciones encargadas del manejo del parseo de mensajes.

- **Encriptación de Mensajes**

Se programa los métodos encargados de realizar la encriptación bajo el algoritmo de Rijnadel, cuya implementación se la tiene en el **Anexo 4.**

- **Ruteo mediante los algoritmos**

Corresponde a la funcionalidad básica que deberá tener el router virtual, aquí se desarrolla lo correspondiente a la búsqueda de caminos hacia un destino determinado, la implementación se la puede apreciar en el **Anexo2.**

4.4 Afinamiento y optimización

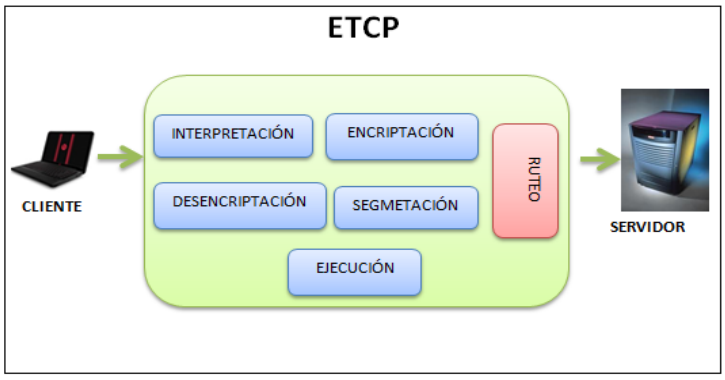
Se implementan cambios de controles en el diseño de las interfaces, como por ejemplo cambio de controles textbox por numericupdown para tener control sobre datos que son exclusivamente numéricos.

En cuanto al proceso de ruteo se propone un algoritmo híbrido entre los algoritmos de Johnson, Dijkstra y Bellman-Ford, lo cual proporcionará una mejor lectura sobre las rutas que deben tomar los paquetes. Se cambia la programación del software para que pueda soportar cualquier dirección IP con su respectiva mascara. Adicionalmente se agrega los conceptos de IP real ya que es necesaria para la comunicación entre computadores.

Importante: El único cuidado que se debe tener para crear instancias del servidor es el de levantar configuraciones en puertos diferentes, ya que los protocolos no permiten más de un escucha de conexión en cada puerto.

A continuación se indica un diagrama de interacción entre componentes externos y componentes del prototipo de protocolo.

Figura 24: Componentes del Sistema

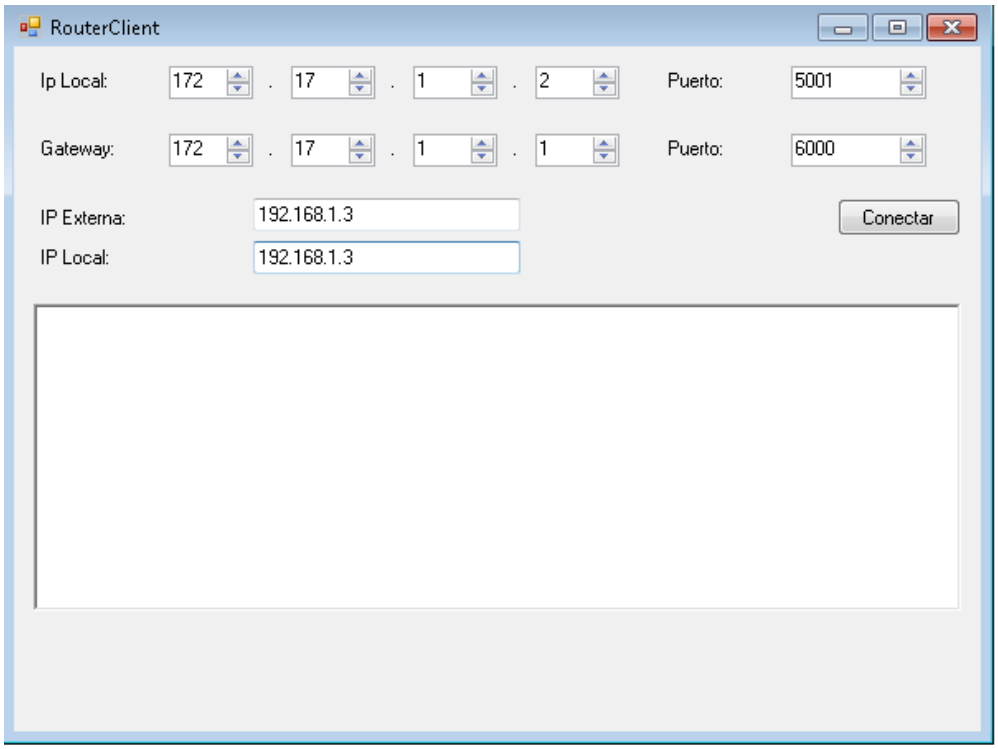


Elaborado por: Jorge Chapaca y David Rojas

De manera parecida al servidor, en el cliente se deben ingresar los datos de la configuración a la que me deseo conectar, siendo muy importantes el puerto, IP local y computadora externa. Para la IP local se debe ingresar la IP con la que se va a conectar el cliente al simulador, más un puerto gateway que es la puerta de enlace de la red de destino a la que se requiere conectar. La computadora externa es la IP real

de donde se está levantado el servidor y finalmente IP local donde está levantado el cliente.

Figura 25: Cliente



Elaborado por: Jorge Chapaca y David Rojas

CAPÍTULO 5
PRUEBAS EXPERIMENTALES

5.1 Caso de estudio

Para el desarrollo del prototipo de protocolo de transporte se tomó como referencia los siguientes aspectos: enrutamiento, confiabilidad, disponibilidad y seguridad. La investigación se basa en el actual problema de la disponibilidad continua de información, junto con los problemas clásicos y comunes que son la confiabilidad, seguridad y el enrutamiento.

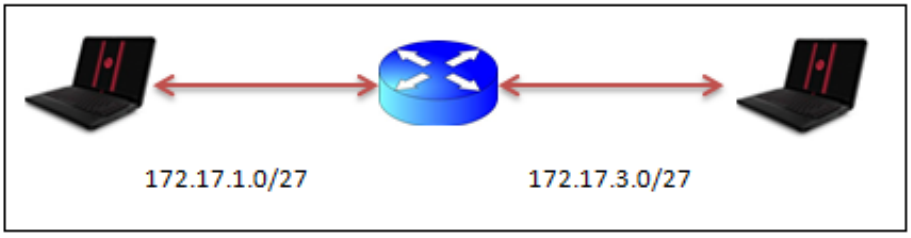
Cuando un usuario de la red se encuentra viendo un video, escuchando una canción o bajando información de la internet es muy común encontrarse con mensajes como: conexión rechazada por el servidor, imposible determinar el host de destino; o que al final de la descarga de un archivo que costó mucho trabajo encontrar faltando pocos segundos o megas para concluir la descarga se pierda la conexión, la causa desconocida para los usuarios que no tienen conocimiento de informática es que se cayó el servidor, causa que es cierta pero no el 100% de los casos, una de la causa puede ser que al traer la información uno de los distintos puntos por donde pasa la información sufrió una avería o simplemente dejo de funcionar, lo que provoca realmente que se pierda la conectividad.

5.2 Topología del caso de estudio

Para probar la problemática descrita en el caso de estudio se han formado tres escenarios:

Escenario 1: Conexión de hosts hacia el router.

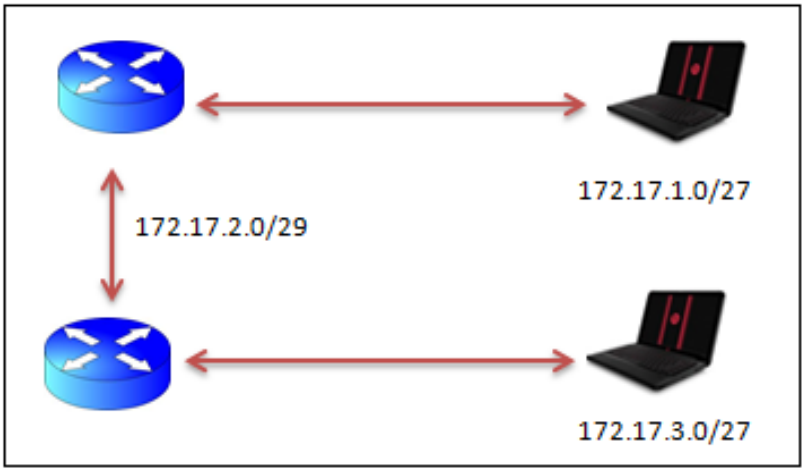
Figura 26: Escenario 1



Elaborado por: Jorge Chapaca y David Rojas

Escenario 2: Conexión entre dos routers y un host por cada uno

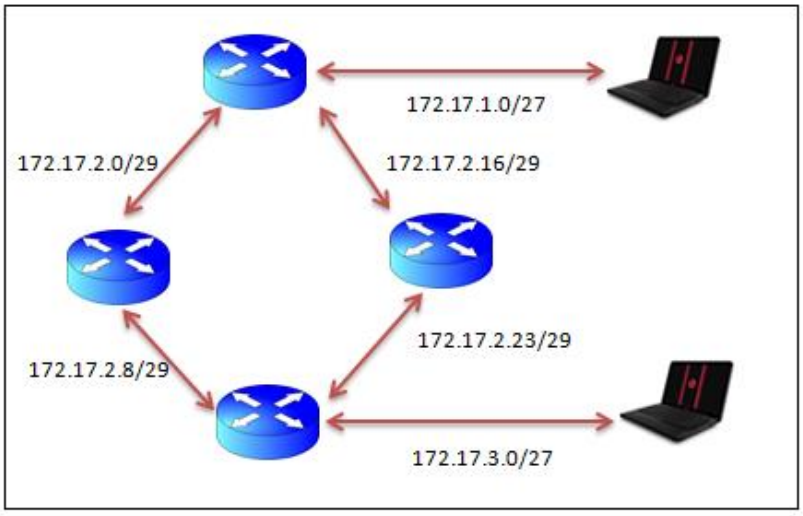
Figura 27: Escenario 2



Elaborado por: Jorge Chapaca y David Rojas

Escenario 3: Conexión de 4 routers, usando dos como puente y 2 hosts conectados entre los dos routers restantes.

Figura 28: Escenario 3



Elaborado por: Jorge Chapaca y David Rojas

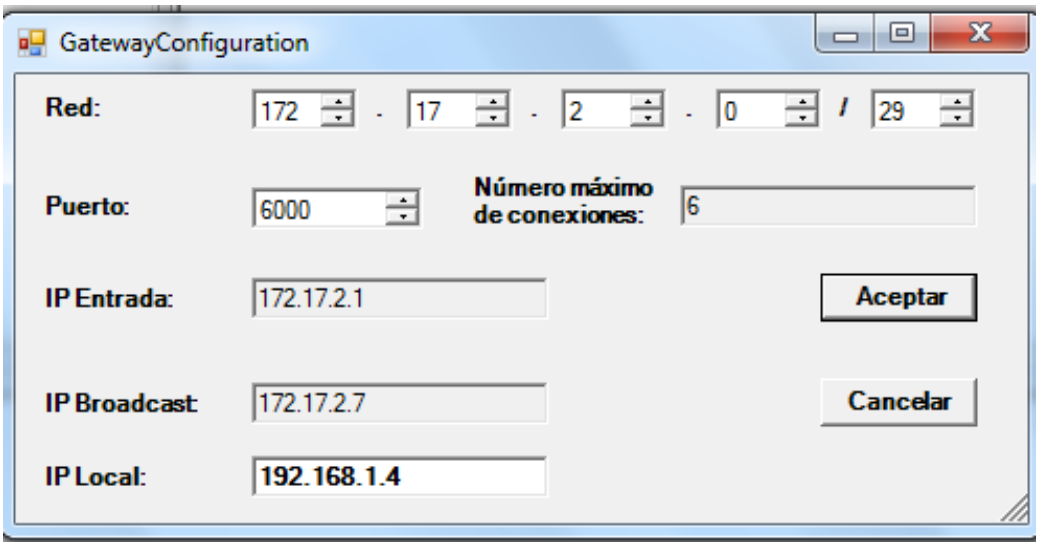
Con la ejecución de estos tres escenarios se obtendrán datos para generar información y poder dar una lectura acerca del rendimiento del nuevo protocolo.

Nota: El diseño de los escenarios se realiza mediante el criterio de pruebas incrementales, es decir partiendo del escenario más simple hacia un escenario complejo. Cabe señalar que no son los únicos escenarios posibles ya que la implementación del prototipo de router virtual está diseñado para soportar cualquier tipo de topología, sin embargo los escenarios propuestos son para poder tomar datos y evaluar resultados; si se requiere realizar otras pruebas de complejidad alta, se pueden agregar cualquier cantidad de routers y hosts, siempre manteniendo los recursos del equipo sobre el cual se ejecutan las diferentes instancias.

5.3 Puesta en marcha del router virtual

Al abrir el ejecutable se selecciona configurar → gateway. En el cuadro de diálogo se ingresan los datos para crear una nueva red, para el ejemplo se creará la red 172.17.2.0/29 en el puerto 6000. Al dar clic en el botón aceptar se llenarán automáticamente los datos correspondientes al número de conexiones disponibles, IP del gateway e IP de broadcast como se muestra a continuación:

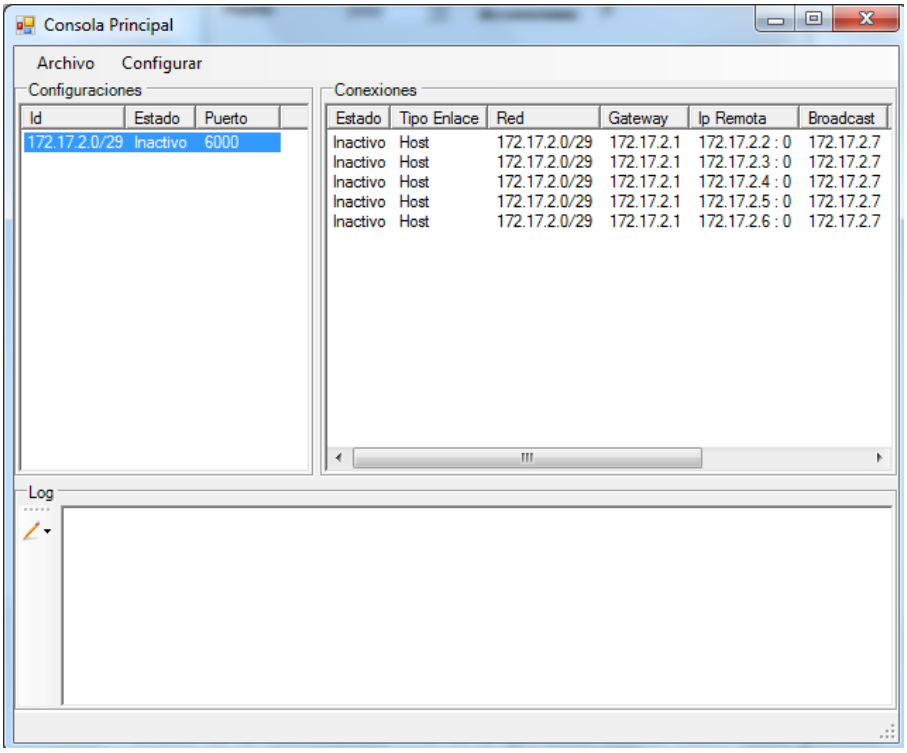
Figura 29: Creación de una red 172.17.2.0/29



Elaborado por: Jorge Chapaca y David Rojas

Después de dar clic en el botón de aceptar inmediatamente se cierra la ventana y se crea la nueva configuración en el router como se muestra en la siguiente figura:

Figura 30: Red 172.17.2.0/29



Elaborado por: Jorge Chapaca y David Rojas

En el panel derecho se muestra en lista todas las posibles conexiones que la configuración soportará. Para arrancar el servidor se da clic derecho sobre la configuración y se selecciona la opción de arrancar.

5.4 Ejecución de procesos

Dentro de la funcionalidad total de la aplicación existen cuatro macro procesos que se ejecutan en todo momento.

Establecer conexión

Se establece la conexión cliente – router; o la conexión router – router, éste proceso es el primer punto en el flujo de comunicación, mediante el mismo se abren las puertas de entrada a las comunicaciones.

Encriptar / Desencriptar mensaje

Es el paso previo al envío o recepción de los mensajes; antes del envío el mensaje es procesado por un algoritmo de encriptación para evitar que terceras personas puedan

descifrar el contenido; al momento de recibir un mensaje inmediatamente se procede a la desencriptación del mismo para que pueda ser interpretado por el sistema.

Envío / Recepción de mensajes

Cuando el canal de comunicación está abierto y el mensaje codificado se envía y se recibe a través del protocolo ETCP por medio del router virtual básico.

Cerrar conexión

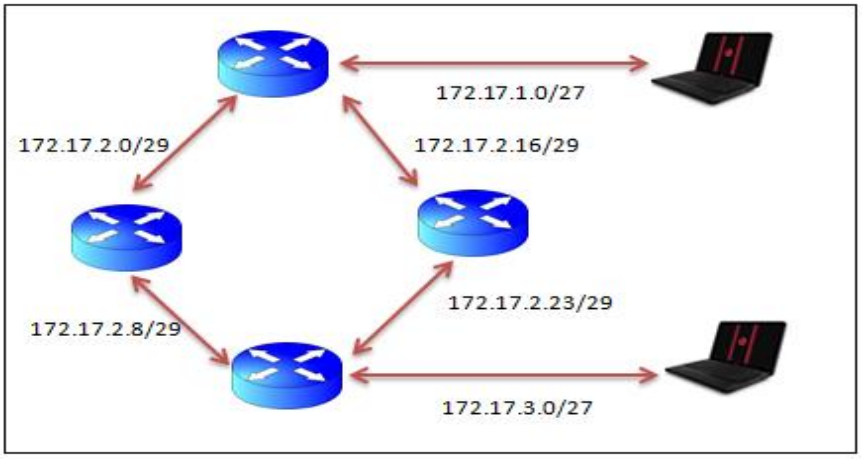
Es el último proceso que se ejecuta, una vez enviado o recibido el mensaje se procede con la liberación del canal de comunicación, el cual sirve para liberar los recursos del sistema, caso contrario se acumularán saturando el sistema.

5.5 Pruebas de funcionamiento

Para la ejecución de pruebas se toma el escenario 3 propuesto en la sección de topologías, que consistirá en conectar cuatro routers entre sí, posteriormente dos host, con el fin de enviar información desde un punto hacia el otro y en el trascurso del envío tratar de simular los posibles errores que se puedan suscitar al momento de realizar el proceso antes mencionado, verificar la caída de algún enlace.

Pérdida de paquetes o que otras personas puedan ver la información que se transmite, por lo que ETCP tiene la finalidad de cumplir aspectos como el enrutamiento, confiabilidad, disponibilidad y seguridad, para tratar de solventar dichos problemas y mejorar la comunicación.

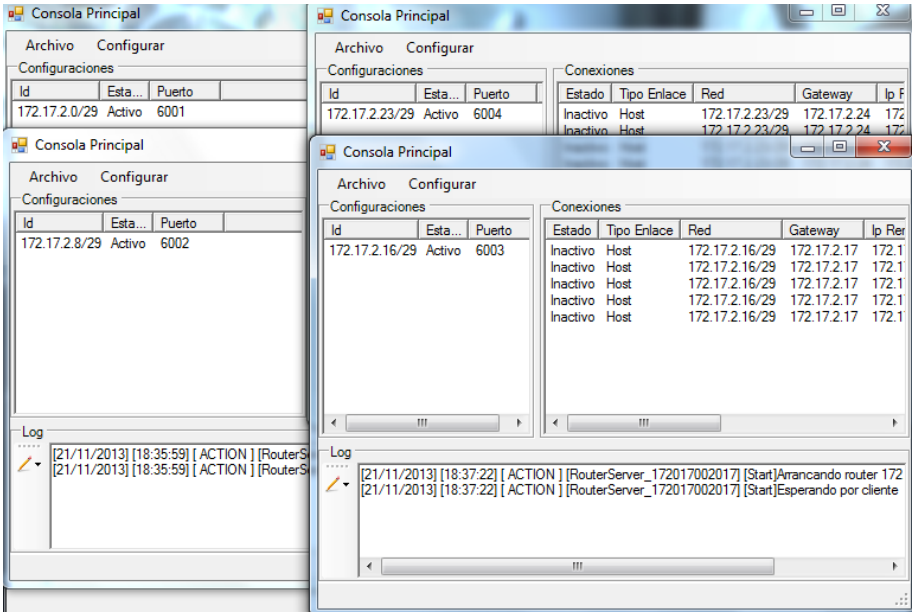
Figura 31: Escenario de Prueba



Elaborado por: Jorge Chapaca y David Rojas

Tomando como referencia los pasos del manual de usuario; ver **Anexo 1**, se configuran las cuatro redes lanzando 4 instancias diferentes de la aplicación, para poder visualizar de mejor manera la interacción entre los componentes, una vez realizados los pasos de configuración se debe arrancar los procesos en los routers para que se abran los canales de comunicación, como se indica en la siguiente figura

Figura 32: Arranque de las cuatro redes



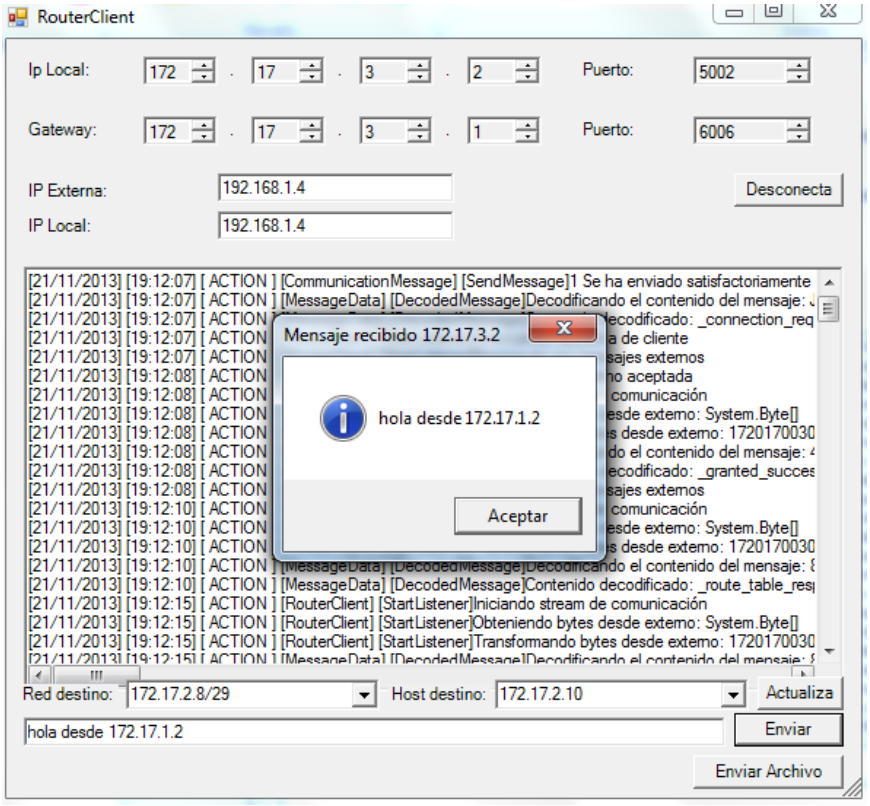
Elaborado por: Jorge Chapaca y David Rojas

Con el arranque de las redes se deberá enlazar como se indica en el escenario 3, éste procedimiento se lo realiza siguiendo las instrucciones del manual de usuario, posteriormente se levantan los hosts 172.17.1.0/27 y 172.17.3.0/27 conectándolos a

las redes correspondientes, de esta manera se obtiene la estructura para realizar las pruebas de funcionamiento.

Con los ambientes configurados y puestos en marcha se procede al envío de un mensaje de texto “hola desde 172.17.3.2”, desde el host 172.17.1.0/27 hacia el 172.17.3.0/27, una vez realizado los pasos anteriores se envía el mensaje y se observará lo siguiente:

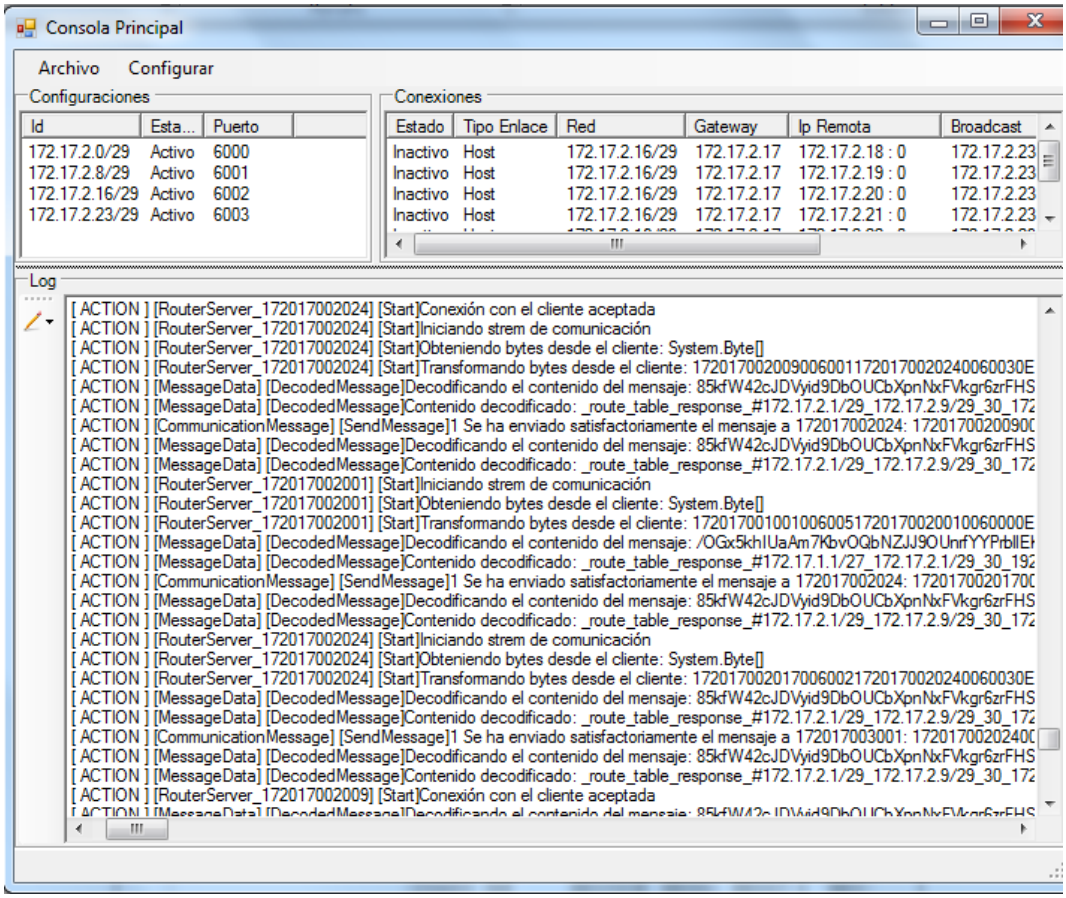
Figura 33: Mensaje recibido en el host 172.17.3.2 desde 172.17.1.2



Elaborado por: Jorge Chapaca y David Rojas

Se verificará cual fue el proceso que realizó el log mediante él envió del mensaje como se muestra a continuación:

Figura 34: Escritura de logs



Elaborado por: Jorge Chapaca y David Rojas

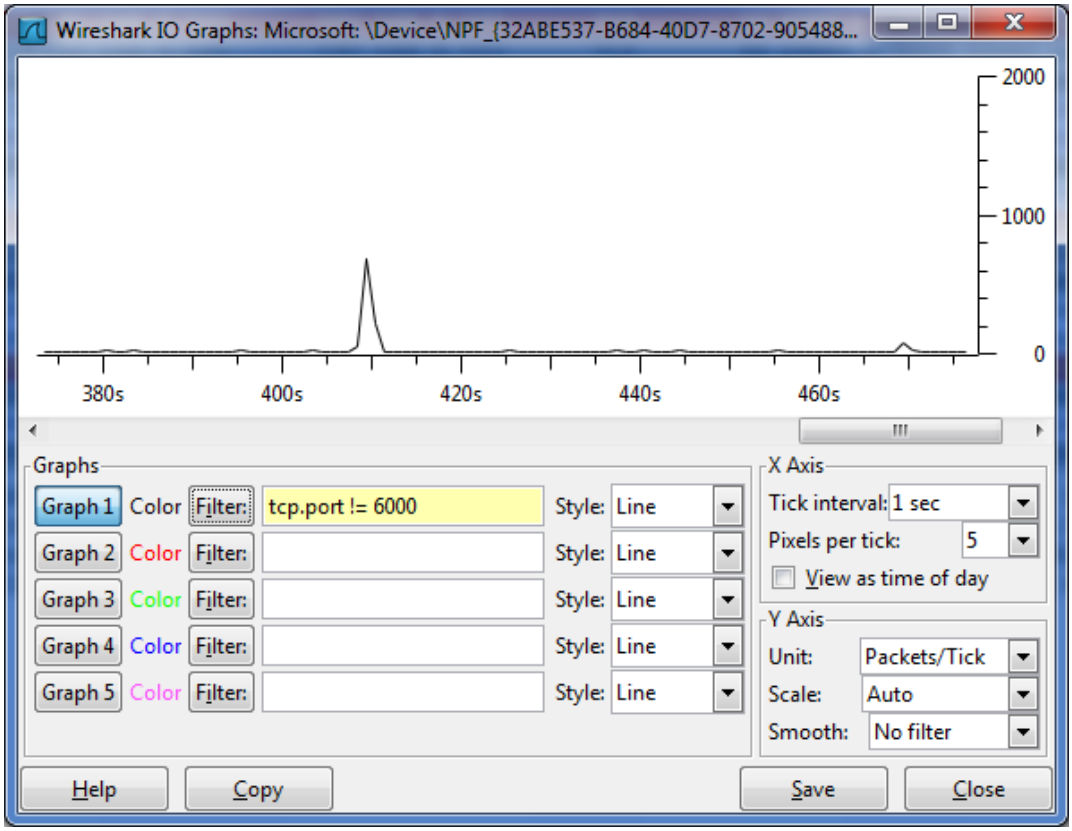
El camino que el mensaje recorrió para que el mensaje llegue desde 172.17.1.2 hacia 172.17.3.2, fué por los routers 172.17.2.0/29 → 172.17.2.16/29 → 172.17.2.23/29; ya que el router 172.17.2.8/29 no registró ninguna actividad durante el envío.

5.6 Estadísticas de las evaluaciones

Para obtener una correcta apreciación de la ejecución de evaluaciones se obtuvo apoyo del software wireshark el mismo que ayuda con la obtención de gráficas estadísticas en la ejecución del sistema.

Se realizó la prueba con una ejecución normal del protocolo TCP, obteniendo los siguientes resultados:

Figura 35: Ejecución de tráfico con el protocolo TCP

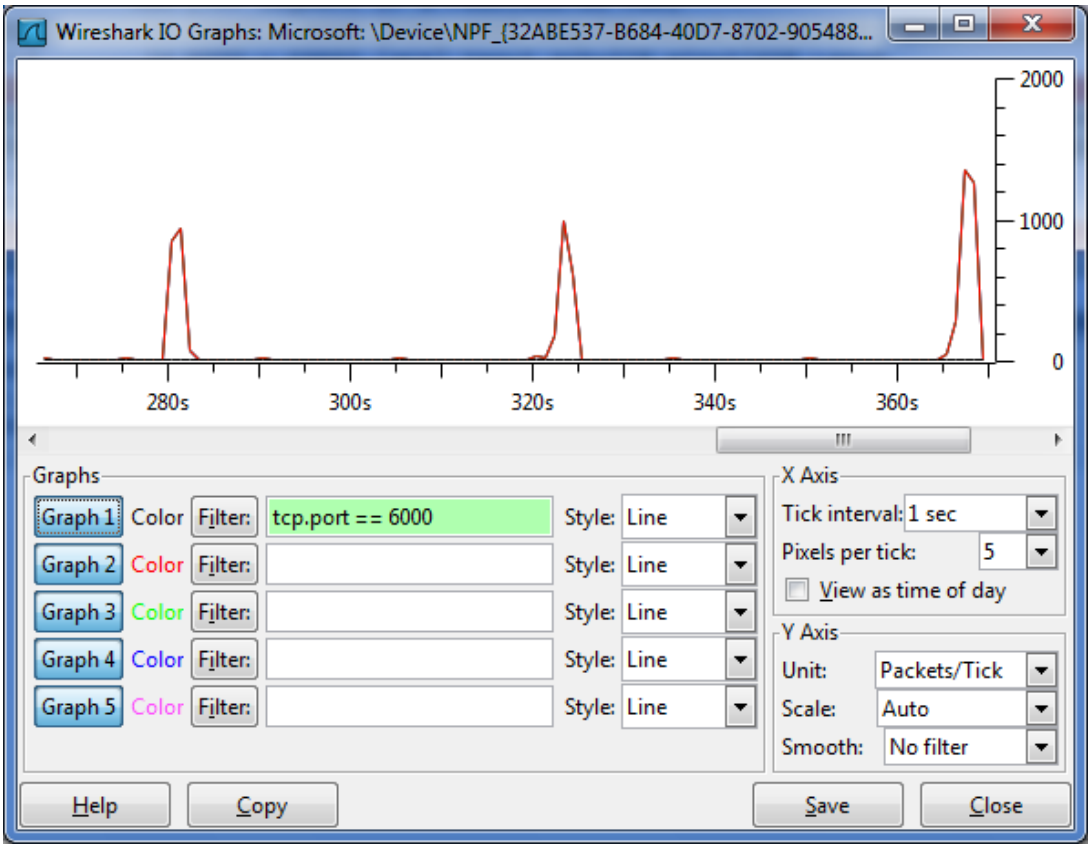


Elaborado por: Jorge Chapaca y David Rojas

El protocolo TCP genera tráfico de entrada - salida de red siempre y cuando se ejecute, lo que lleva a la conclusión de que solo se abre cuando se solicita.

Con el nuevo prototipo de protocolo se observa un funcionamiento muy similar, obteniendo los picos de transmisión en los momentos en los que fue requerido, para esto se ejecutó 3 veces una nueva transmisión de datos, lo que corresponde a los 3 picos que aparecen en el gráfico a continuación.

Figura 36: Ejecución de tráfico con el nuevo protocolo, utilizando el software wireshark



Elaborado por: Jorge Chapaca y David Rojas

5.7 Análisis de resultados obtenidos

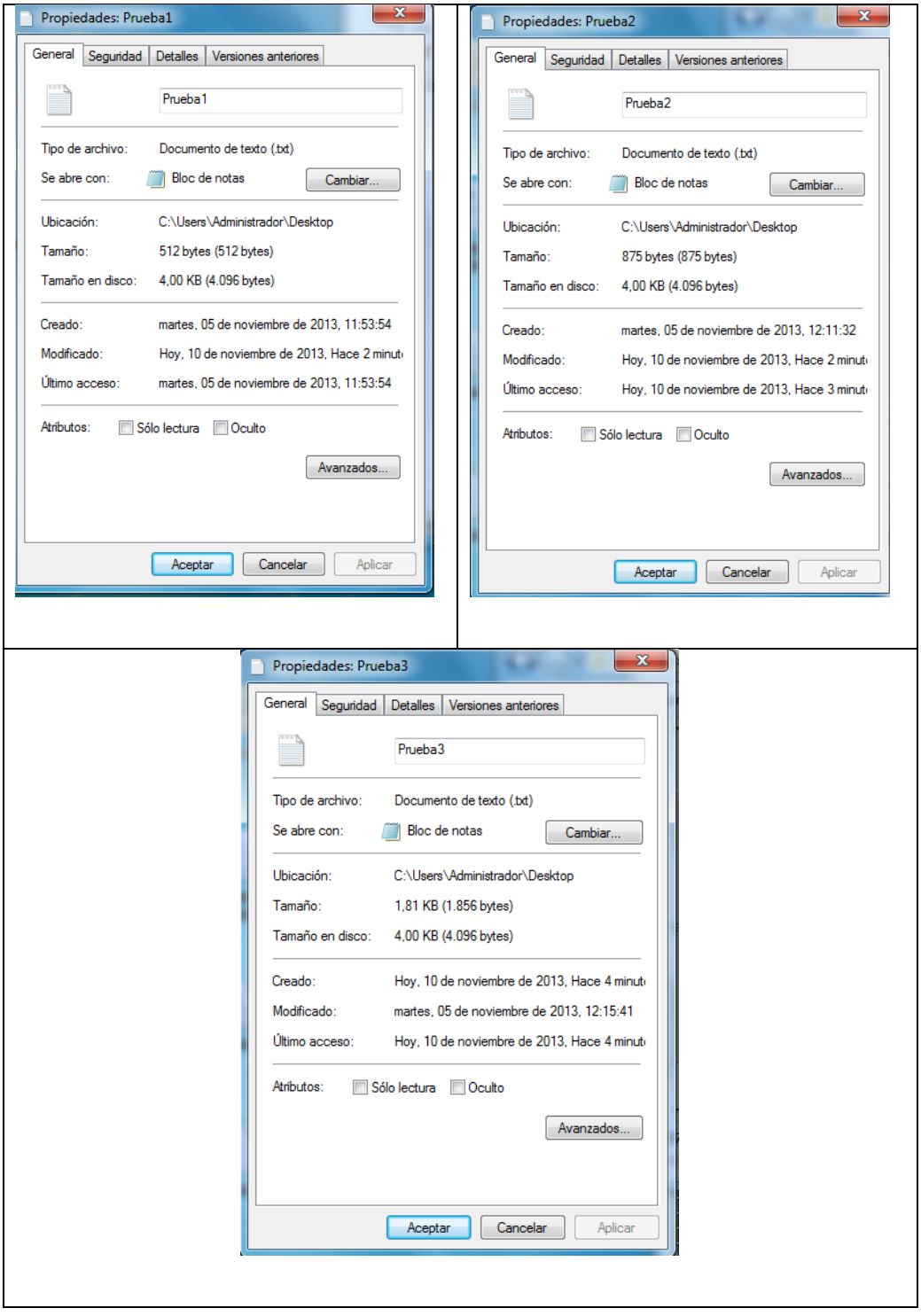
Para el análisis de resultados obtenidos se tomaron en cuenta tres puntos fundamentales: confiabilidad, disponibilidad y encriptación.

Confiabilidad

ETCP permitirá asegurar la entrega confiable de los datos desde su punto de origen hacia su punto de llegada asegurándose que siempre la información llegue de manera constante se encuentre disponible; para esto se realizó las debidas pruebas que se mostrarán a continuación.

Se procederá a realizar el envio de tres archivos de texto plano con el fin de diagnosticar que sucede en el transcurso del envio de la información como se muestra en la Figura 37.

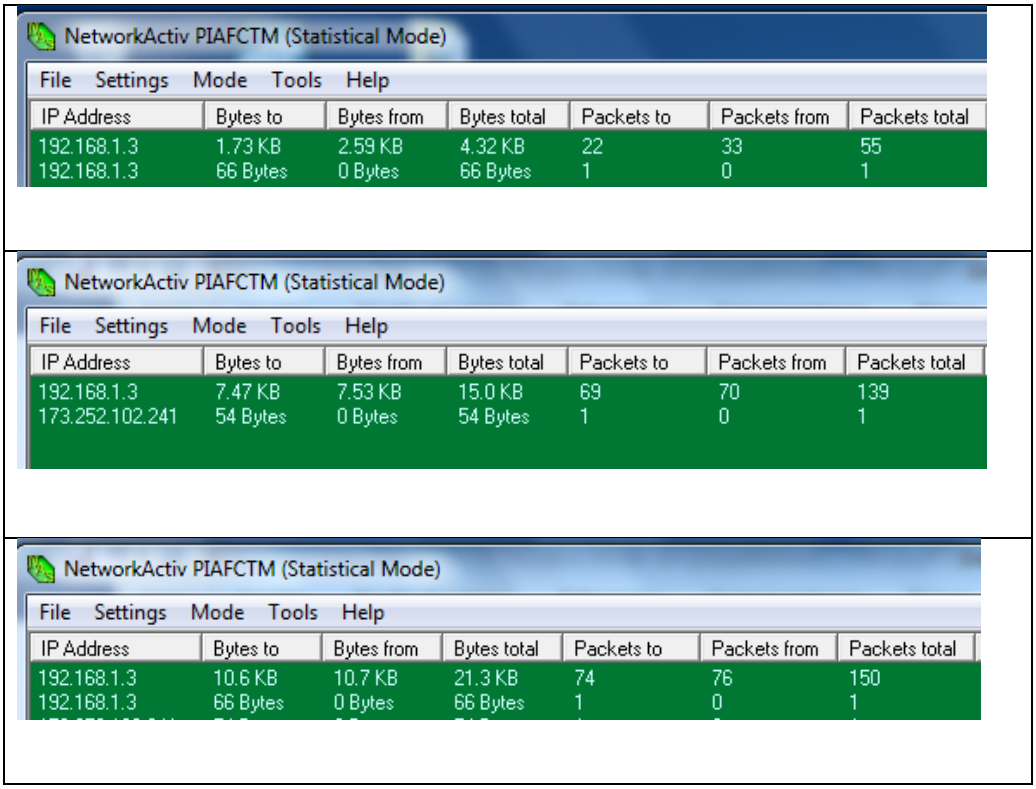
Figura 37: Envió de archivo plano origen hacia destino



Elaborado por: Jorge Chapaca y David Rojas

Como se observa en la herramienta utilizada "networkactiv", que permitirá ver el proceso que realizó el prototipo de protocolo de transporte creado al momento de enviar un archivo plano, considerando que no existe pérdida en los paquetes como se muestra a continuación.

Figura 38: Envió de archivo plano origen hacia destino

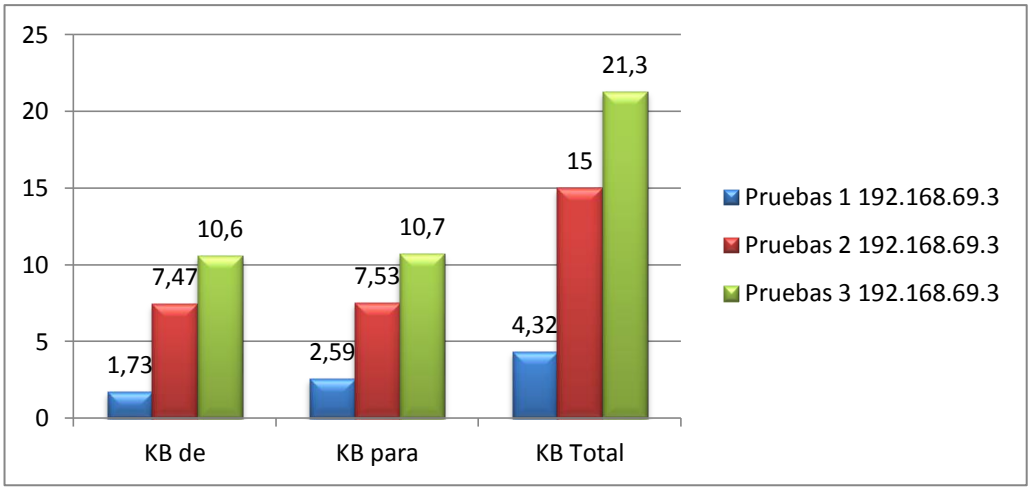


Elaborado por: Jorge Chapaca y David Rojas

Se tomará los datos reflejados en la **Figura 38** para observar estadísticamente que no existe pérdida alguna en cada proceso de envío, cabe recalcar que el prototipo de protocolo de transporte; realiza pasos anteriores como enrutar, encriptar, desencriptar y finalmente segmentar. Por tal motivo serán más bytes los que se reciban en el destino para poder ensamblar de manera confiable y exitosa.

Figura 39: Análisis de los paquetes procesados.

Pruebas	Direccion IP	KB de	KB para	KB Total
Pruebas 1	192.168.69.3	1,73	2,59	4,32
Pruebas 2	192.168.69.3	7,47	7,53	15
Pruebas 3	192.168.69.3	10,6	10,7	21,3

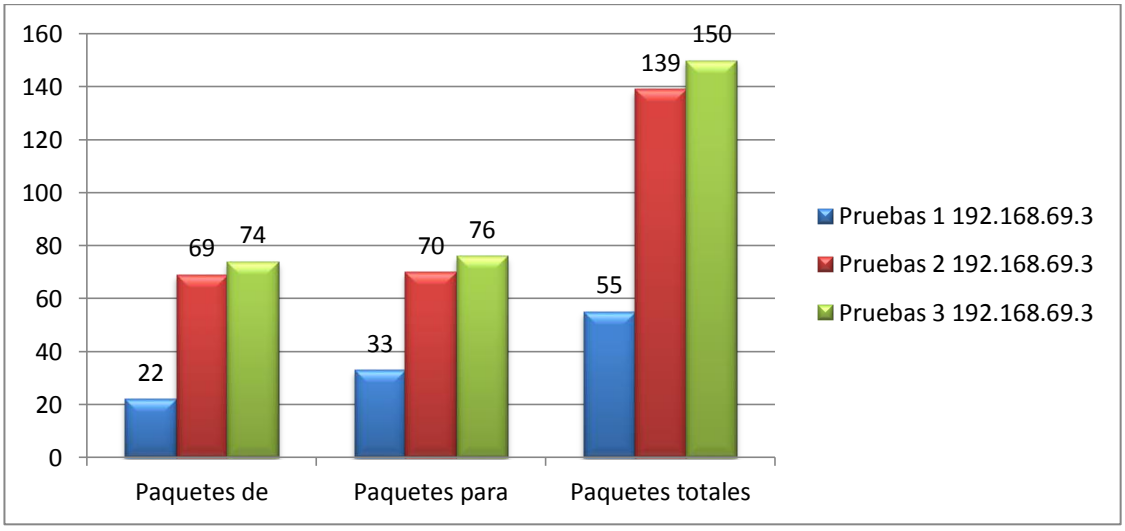


Elaborado por: Jorge Chapaca y David Rojas

De igual forma se realiza un análisis de los paquetes que interactuaron en el proceso del envío del archivo plano desde su origen hacia el destino, obteniendo como conclusión que los paquetes fueron enviados y llegaron de manera confiable y exitosa, verificando que no existen paquetes perdidos durante el proceso de transmisión como se muestra en la **Figura 39.**

Figura 40: Análisis de Paquetes

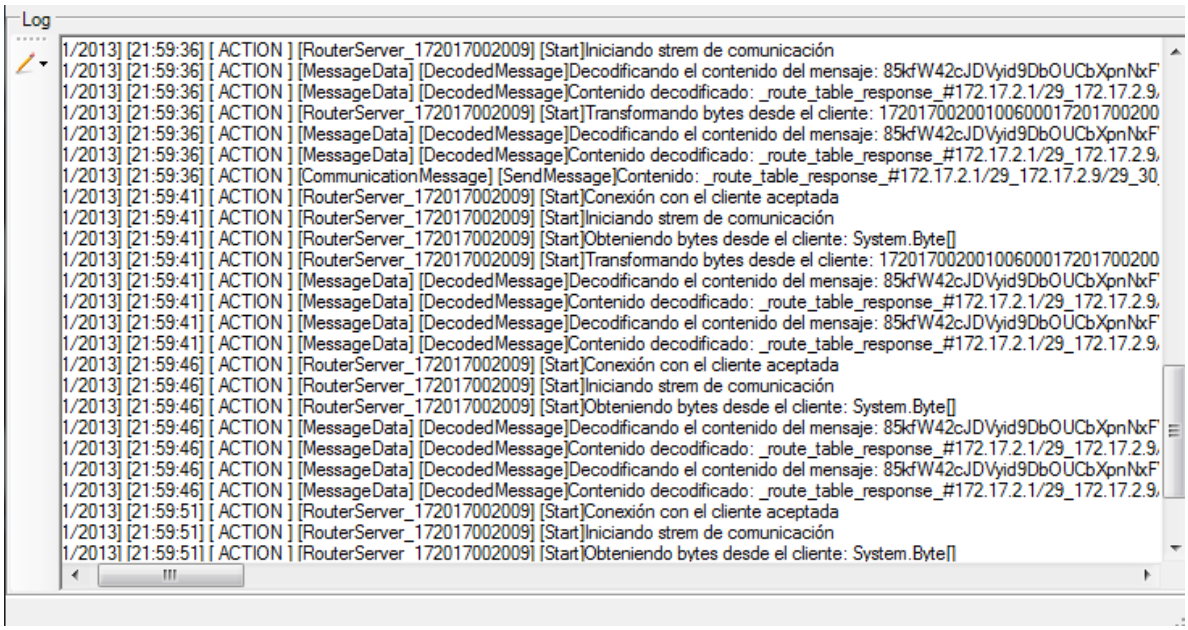
Pruebas	Direccion IP	Paquetes de	Paquetes para	Paquetes totales
Pruebas 1	192.168.69.3	22	33	55
Pruebas 2	192.168.69.3	69	70	139
Pruebas 3	192.168.69.3	74	76	150



Elaborado por: Jorge Chapaca y David Rojas

En la siguiente imagen se podrá observar el log del servidor que indicará el proceso que se realizó durante la transmisión para que el archivo llegue a su destino de manera confiable y exitosa.

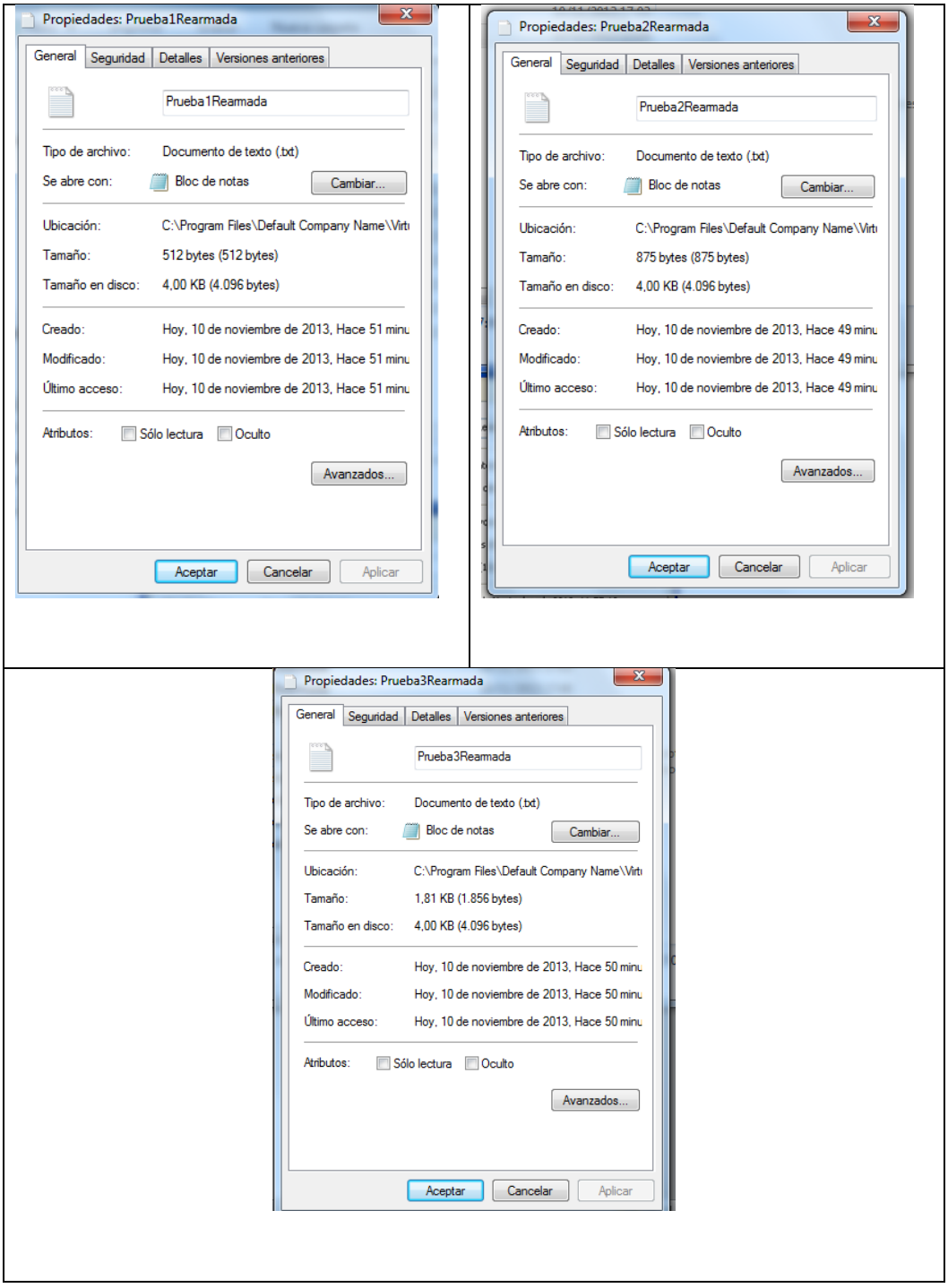
Figura 41: Logs del servidor al realizar el proceso de envió de archivo



Elaborado por: Jorge Chapaca y David Rojas

En el log se puede apreciar que lo primero es verificar si no existe algún problema para establecer la comunicación desde un punto origen hacia un punto destino, una vez confirmado esto se realiza un ruteo con el fin de conocer cuál es la ruta más corta a ser tomada, de esta manera la información es enviada de forma confiable para que la misma llegue hacia el destino. Cabe recalcar que la verificación de la comunicación desde un punto hacia otro siempre va estar activa, a menos que se finalice la conexión.

Figura 42: Rearmada del archivo y su coste



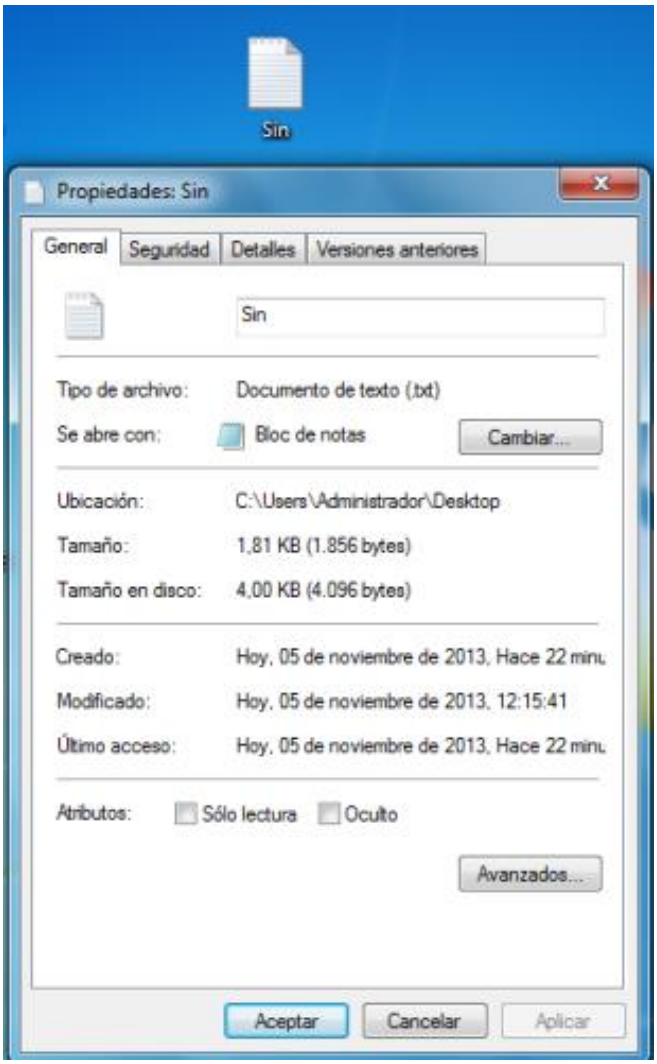
Elaborado por: Jorge Chapaca y David Rojas

Disponibilidad

ETCP protege de interrupciones o caídas de forma automática y en un corto plazo de tiempo con el fin de mantener siempre disponibilidad para la transferencia de datos.

Para esto de igual manera se realiza las pruebas respetivas ver **Figura 28:** Escenario de Prueba, a continuación se va a proceder a enviar un archivo con el nombre.

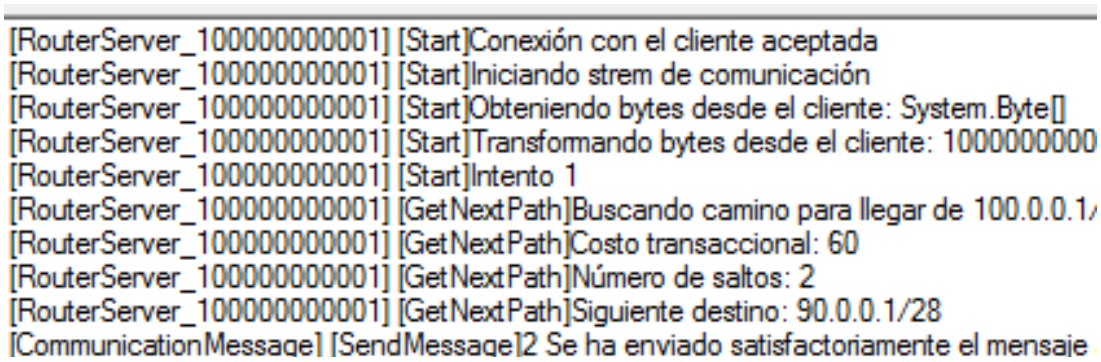
Figura 43: Prueba Disponibilidad archivo Sin



Elaborado por: Jorge Chapaca y David Rojas

En el siguiente gráfico se puede observar en el log cuál fue el camino optado para él envió del archivo.

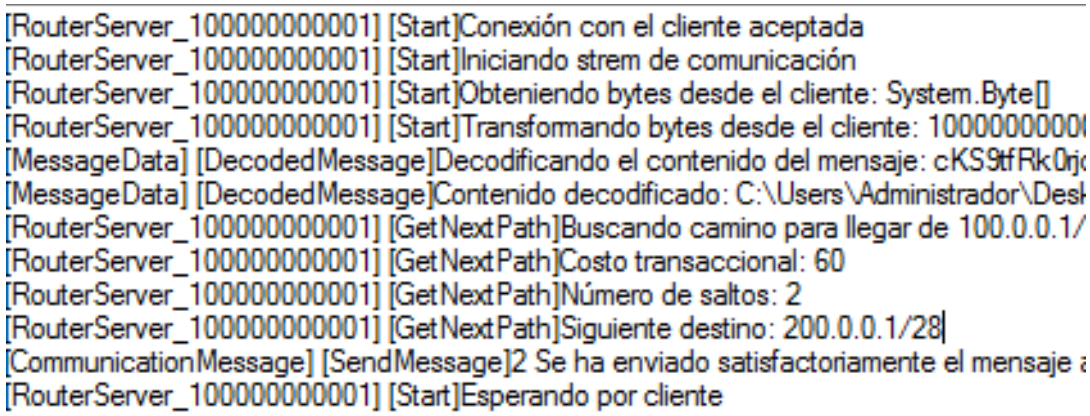
Figura 44: Prueba Disponibilidad



Elaborado por: Jorge Chapaca y David Rojas

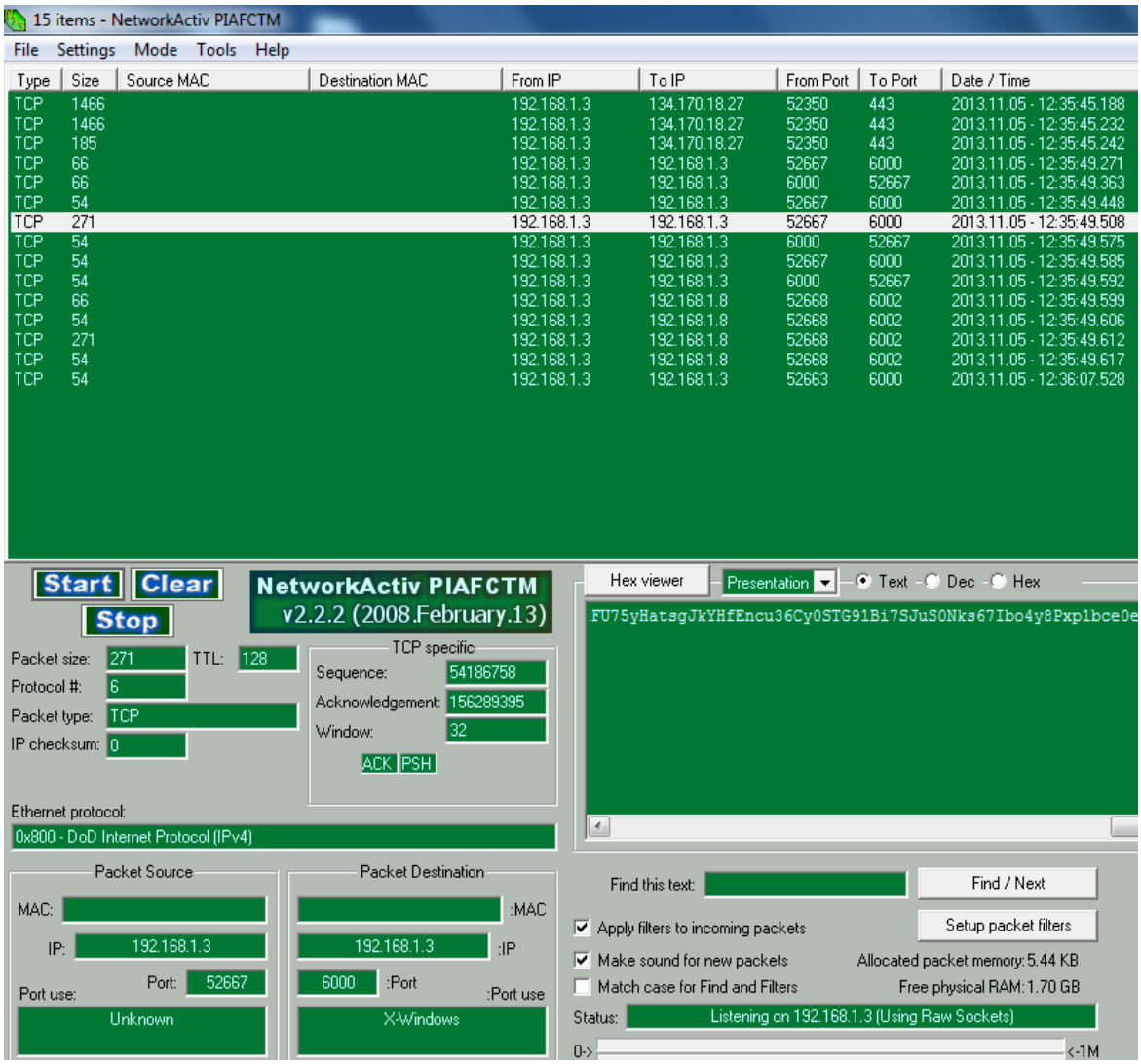
Ahora para la prueba de disponibilidad se volverá a enviar el archivo pero en el transcurso del envío se detendrá el router por donde se está enviando el archivo que sería por la red 172.17.2.8/29 dando un escenario que ocurre en nuestros medios o en la comunicación que se conoce como caída de un enlace, aquí se aplica la definición de alta disponibilidad, además verificar en que tiempo se lograría solucionar este inconveniente, el camino escogido inmediatamente por la red fue 172.17.2.23/29, como se indica en la **Figura 44**.

Figura 45: Prueba Disponibilidad



Elaborado por: Jorge Chapaca y David Rojas

Con la herramienta que se ha trabajado en el transcurso de las pruebas, se puede verificar como viaja por la red el archivo que se envió para la prueba de alta disponibilidad y el cambio que este realizó cuando se cerró el router con la red 172.17.2.8/29 y optó por cambiarse a la red 172.17.2.23/29 para terminar de manera exitosa la transmisión.



Elaborado por: Jorge Chapaca y David Rojas

Cuando se realizaron las pruebas respectivas se establecieron dos rutas para llegar a un mismo punto y en medio de la transmisión de los datos se desconectó una de las rutas, se pudo apreciar en los logs de los servidores que inmediatamente se tomaba el camino alterno para la ejecución, teniendo de esta manera la comunicación siempre disponible.

En primera instancia el sistema muestra errores al momento de desconectar alguno de los terminales, sin embargo se mitigó esos fallos con la implementación de una bandera de retorno al momento de enviar los mensajes, para este fin se modificó la clase communicationmessage (**Anexo 3**) para que el tipo de retorno al mandar un mensaje sea un verdadero o falso dependiendo si se envió o no la data al otro

extremo de la comunicación, finalmente se realiza una serie de tres intentos para enviar un mensaje, para tal efecto se puede revisar el **Anexo 2** correspondiente a la implementación base del protocolo ETCP.

Usando los resultados obtenidos se pudo apreciar que siempre que exista un camino para llegar al destino se entregará el mensaje a su receptor, ya que ETCP es capaz y está preparado siempre para un escenario donde se corte la comunicación de alguno de sus nodos.

Encriptación

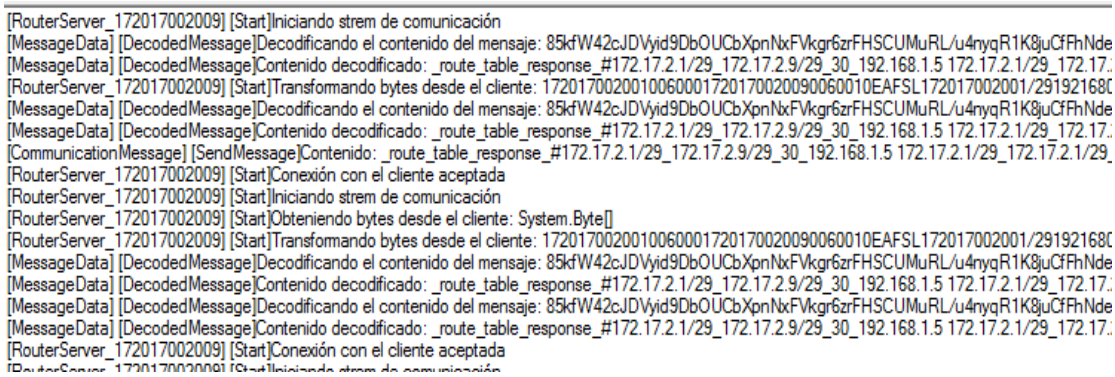
En las pruebas realizadas se probó el nivel de encriptación de los mensajes, concepto que va de la mano con la seguridad, en este punto se pudo lograr el cometido principal ya que la lectura del contenido del mensaje era difícil de descifrar, cabe recalcar que en este punto no existe algoritmos empleados que den el 100% de seguridad al momento de encriptar la información, pero en el cometido se puede asegurar que cumple con la funcionalidad de no dar a conocer a cualquiera el contenido de los mensajes enviados por la red.

En segunda instancia se observará la encriptación que se realiza en cada proceso con el fin de que estos mensajes viajen por la red y no puedan ser vulnerados por personas ajenas. Para ejemplo se tomó la figura 28: Escenario de Prueba

Se levanta la red con la siguiente dirección IP 172.17.2.0/29.

Se obtiene como resultado la decodificación del contenido / codificación del contenido como se muestra en el log.

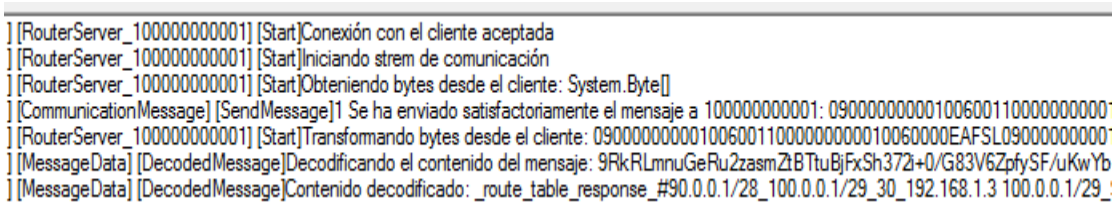
Figura 47: Log de contenidos



Elaborado por: Jorge Chapaca y David Rojas

De igual manera se levanta la red con la siguiente dirección 172.17.2.8/29.

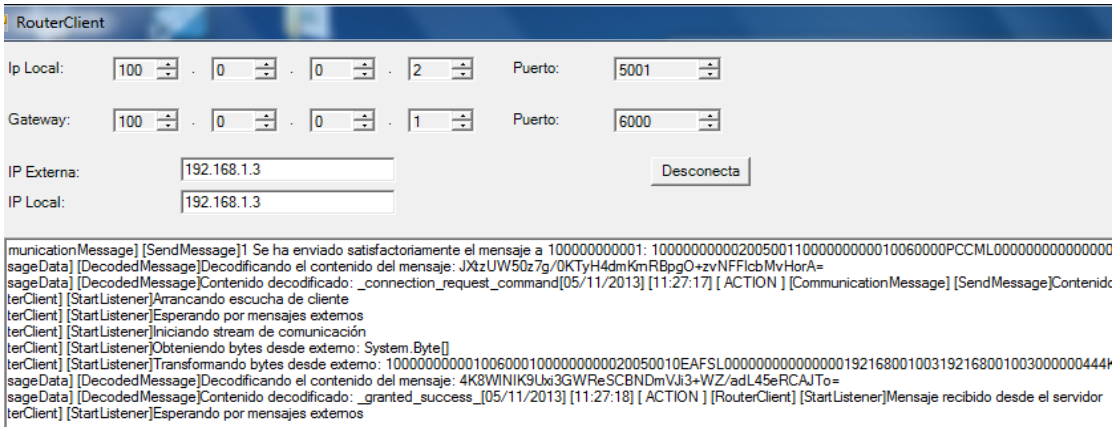
Figura 48: Log de



Elaborado por: Jorge Chapaca y David Rojas

Y finalmente para el cliente sería el host con la IP 172.17.1.2.

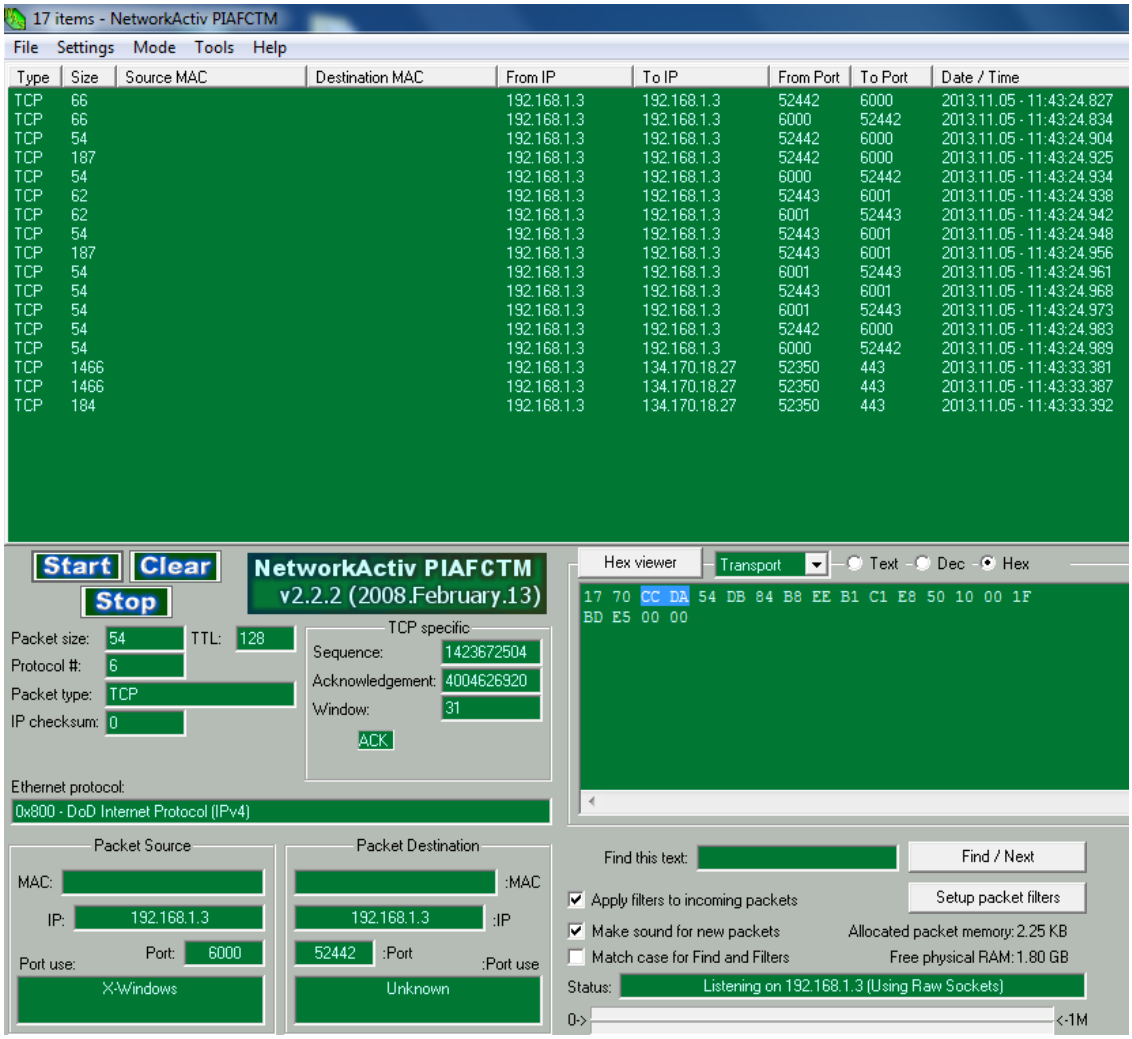
Figura 49: Herramienta cambios



Elaborado por: Jorge Chapaca y David Rojas

Para un mejor análisis en tiempo real se optó por utilizar la herramienta "networkactiv" que permitirá monitorear lo que está realizando el sistema.

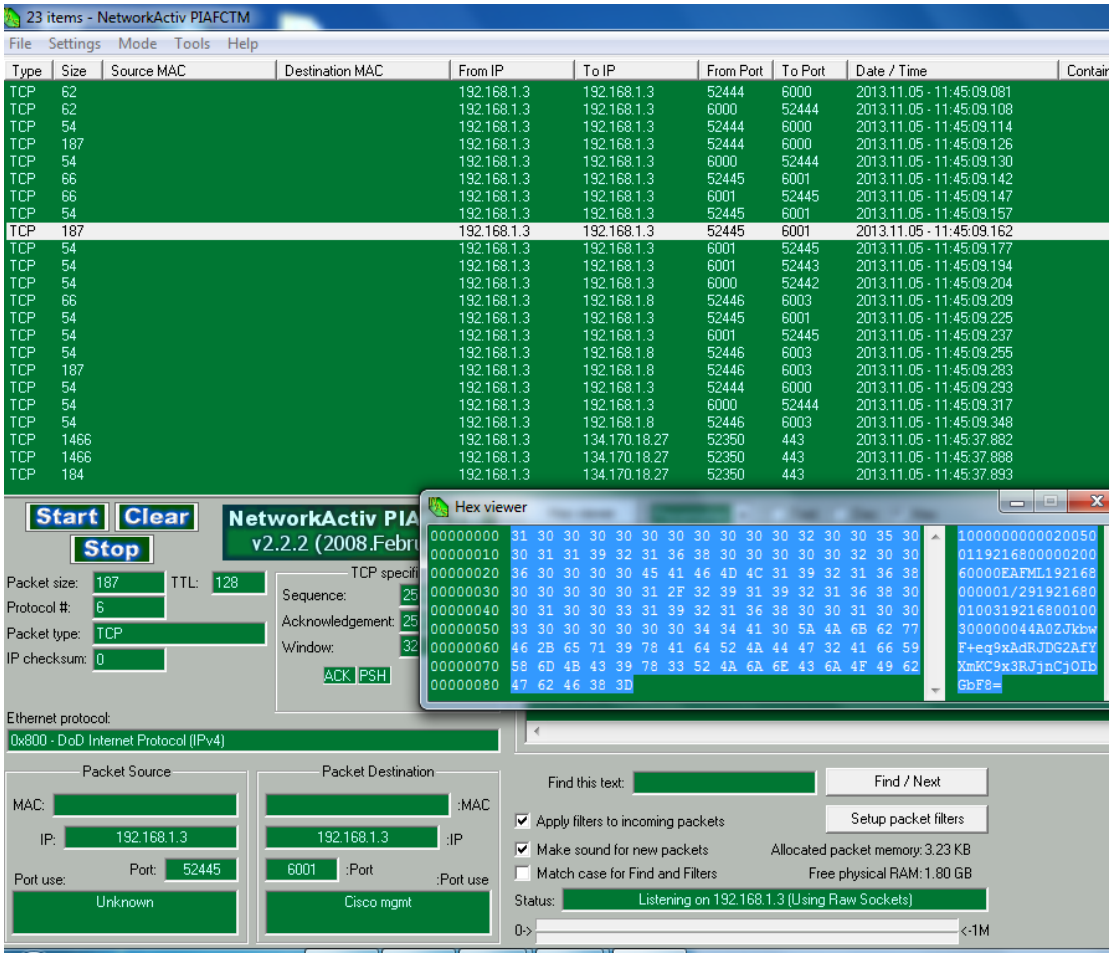
Figura 50: Herramienta cambios



Elaborado por: Jorge Chapaca y David Rojas

Una vez que se empieza a realizar las pruebas de lo que está viajando por la red se puede verificar, y como es de esperarse el mensaje viaja encriptado con el fin de asegurar la información para que no pueda ser violentada durante el envío desde el origen hacia el destino; como se puede verificar en la gráfica.

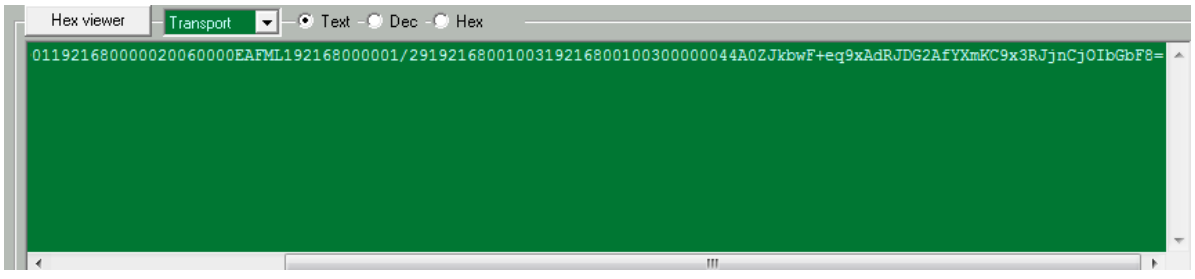
Figura 51: Herramienta cambios



Elaborado por: Jorge Chapaca y David Rojas

De igual manera en la siguiente gráfica se podrá verificar lo que se transporta y cómo se realiza una encriptación que sea confiable y no pueda ser fácil para personas indelicadas que deseen acceder.

Figura 52: Herramienta cambios

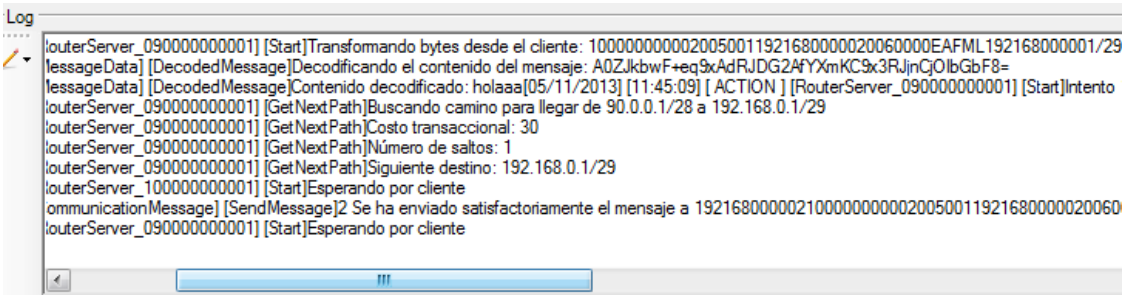


Elaborado por: Jorge Chapaca y David Rojas

En este gráfico se puede observar la verificación del logs del servidor para corroborar la información de la herramienta "networkactiv" al momento de cómo viaja la

encriptación por la red; como se puede ver durante el proceso descrito a continuación.

Figura 53: Herramienta cambios



Elaborado por: Jorge Chapaca y David Rojas

Al principio el manejo del concepto de encriptación causó problemas en el envío de paquetes porque se generan caracteres extraños, por lo que se realizó una encriptación especial, la misma que consiste en encapsular los paquetes del archivo y posteriormente partirlos en secciones, de esta forma se obtuvo una data más limpia al momento de enviar los paquetes, la clase que implementa esta funcionalidad se encuentra detallada en el **Anexo 6.**

El mismo aspecto se dio al realizar el parseo de los datos, ya que al momento de enviar los mensajes se realiza un rellenado de bytes, así que fue necesario incrementar una bandera que indique la longitud original del contenido del mensaje, ésta implementación se encuentra en el **Anexo 5.**

5.8 Especificación del protocolo vs TCP

La siguiente tabla muestra las similitudes y diferencias entre el protocolo creado y TCP.

Nota: La tabla muestra las ventajas en color celeste, desventajas en color tomate, las igualdades en color blanco y en color gris los parámetros que pueden ser considerados ventaja o desventaja dependiendo del contexto.

Tabla 12: Especificación del protocolo vs TCP

TCP	**ETCP**
Conexión	
Orientado a la conexión.	
No puede concluir una transmisión sin todos los datos en movimiento explícitamente confirmados.	Puede trabajar en modo que no se necesite la confirmación de recepción de los datos enviados para terminar una transmisión.
Mensajes (Si es una transacción donde viaja información sensible del usuario es necesaria la confiabilidad, pero si es una descarga de un archivo es necesario disponibilidad).	
Confiabilidad y seguridad en la entrega.	Confiabilidad y seguridad en la entrega.
División de mensajes en datagramas.	División del contenido del mensaje.
Seguimiento de orden.	Soporte para recepción en desorden.
Usa checksums para la detección de errores.	No implementa detección de errores.
Prioridad (Si es una transacción donde viaja información sensible del usuario es necesaria la confiabilidad, pero si es una descarga de un archivo es necesario disponibilidad).	
Confiabilidad y seguridad en la entrega.	Confiabilidad y seguridad en la entrega.
Recursos	
Los recursos son manejados por el sistema operativo.	
Garantía	
• Que los datos lleguen. • Lleguen en orden. • Lleguen sin duplicados.	• Que los datos lleguen. • No importa el orden. • No importa si existen duplicados.

Elaborado por: Jorge Chapaca y David Rojas

5.9 Manual de usuario

Para el presente proyecto se desarrolló un manual de usuario, en el cual consta la instalación y la utilización del software que muestra el funcionamiento del prototipo de protocolo ETCP paso a paso.

(VER ANEXO No. 1)

CONCLUSIONES

- Los protocolos de comunicación existentes ofrecen diferentes características de seguridad, confiabilidad y disponibilidad, sin embargo del estudio realizado se ve que existe la posibilidad de proponer mejoras e implementar un nuevo prototipo de protocolo de transporte como es ETCP.

- ETCP está basado en el actual protocolo TCP por lo que el resultado obtenido es el de brindar mejoras de seguridad, confiabilidad y alta disponibilidad en cada proceso de comunicación que se realiza en comparación al actual protocolo TCP.

- El Router Virtual realiza una interacción constante de los terminales de comunicación, para obtener de esta manera siempre la información actualizada en lo que se refiere a tablas de ruteo en cada punto.

- El algoritmo que asegura la alta disponibilidad se creó en el transcurso del desarrollo del trabajo investigativo, siendo este implementado por los investigadores, el mismo que mantiene siempre la información actualizada y disponible para el cliente final.

- Del análisis de resultados se comprobó que existe seguridad en el transporte de la información, ya que el mensaje es encriptado y pasa por cada punto de la red, siendo este desencriptado solo en el destino final de la comunicación usando el algoritmo de Rijndael.

- De las pruebas generadas se logró comprobar la confiabilidad en el transporte de la información, ya que el mensaje que se envíe llega a su destino en su condición original.

- Durante la ejecución de las pruebas se pudo determinar cambios considerables en la implementación del nuevo prototipo de protocolo ETCP, una de ellas fue agregar identificadores y banderas en los parámetros de cabecera de mensajes, para que de esta manera sea capaz de soportar la

llegada de los datos de manera desordenada y posteriormente estos datos sean reemsamblados conforme al orden en el que lleguen, para finalmente realizar la segmentación en el traspaso de la información entre los distintos puntos de la red

- El nuevo prototipo de protocolo ETCP sigue utilizando los parámetros de conexión que usan los demás protocolos, estos son una dirección IP y un puerto, de esta manera se asegura la comunicación con el resto de protocolos.

RECOMENDACIONES

- Buscar una secuencia de puertos que no hayan sido usados aún, de esa manera la aplicación desarrollada creara los enlaces directamente sin necesidad de preguntar el puerto donde se levantara.

- Implementar un modelo de programación multi-hilo, para que de esta manera la aplicación sea menos vulnerable a fallos que se pueden presentar durante el proceso de envío de información.

- Usar la tecnología WPF de Microsoft para mejorar la funcionalidad del prototipo propuesto ya que este contiene librerías las cuales soportan la concurrencia en la transmisión de datos.

- Ejecutar la aplicación una por máquina física, para de este modo evitar el bloqueo de los hilos de ejecución y un posible choque por el uso de los puertos, haciendo que se obtenga una mejor lectura de resultados.

- A pesar que se han seleccionado ya los algoritmos de ruteo, la arquitectura propuesta deja abierta la posibilidad de mejoras o cambios, por lo que se recomienda investigar más a fondo los demás algoritmos.

- Se recomienda no utilizar en las pruebas con la aplicación prefijos menores a /26, por que debido a la falta de recursos en el simulador este podría tener problemas al momento de ejecutar los procesos debido al listado de número de host que se generan.

- Para optimizar la aplicación se recomienda añadir un módulo que genere el monitoreo en tiempo real de los procesos que realiza ETCP, con el fin de tener cuadros estadísticos y no depender de otros software para realizar el análisis de resultados.

LISTA DE REFERENCIA

- **BELLMAN,** R. E., (2001), On a routing problem, quarterly of *Applied Mathematics*, 16, 87-90.

- **DONALD** B. Johnson. (January 2000), Algoritmo Johnson, Efficient algorithms for shortest paths in sparse networks. Journal of the ACM, 24(1):1-13.

- **YAO** – Wen, (2004), **Algoritmo Johnson,** Algorithms Problems for shortest path. Taiwan University.

- Arodrigu.webs, (2008), Algoritmos de bellman ford; Recuperado 05 de septiembre de 2012 de:

  http://arodrigu.webs.upv.es/grafos/doku.php?id=algoritmo_bellman_ford

- Jariasf.wordpress, (2013), Algoritmos de bellman ford; Recuperado 18 de enero de 2013 de:
  http://jariasf.wordpress.com/2013/01/01/camino-mas-corto-algoritmo-de-bellman-ford/

- Jariasf.wordpress, (2012), Caminos de algoritmos de dijkstra; Recuperado 20 de abril de 2013 de:
  http://jariasf.wordpress.com/2012/03/19/camino-mas-corto-algoritmo-de-dijkstra/

- Bioinfo.uib.es, (2001), Definición algoritmos de dijkstra; Recuperado 13 de octubre de 2012 de:
  http://bioinfo.uib.es/~joemiro/aenui/procJenui/ProcWeb/actas2001/saalg223.pdf

- Vteforte, (2008), Formato del datagrama UDP; Recuperado 06 de noviembre de 2012 de: http://vteforte.tripod.com/tcp.htm

- Tools.ietf.org, (2010), Formato del datagrama UDP; Recuperado 03 de febrero de 2013 de: http://tools.ietf.org/html/rfc768

- Laynetworks.com, (2002), Los parámetros de medición y comparación de TCP; Recuperado 21 de marzo de 2013 de: http://www.laynetworks.com/Comparative%20analysis_TCP%20Vs%20UDP.htm

- Vteforte, (2000), Protocolos de transporte; Recuperado 21 de marzo de 2013 de: http://vteforte.tripod.com/tcp.htm

- Neo.lcc.uma, (2003), Protocolos de transporte; Recuperado 03 de febrero de 2013 de: http://neo.lcc.uma.es/evirtual/cdd/tutorial/transporte/protrans.html

- Inf.uct.cl, (2004), Protocolos de transporte; Recuperado 21 de marzo de 2013 de: http://www.inf.uct.cl/~amellado/archivos/teleprocesos2.pdf

- Wireshark.org, (2003), Software wireshark; Recuperado 11 mayo de 2013 de: http://www.wireshark.org/

- Networkactiv.com, (2010), Software NetworkActivPiafctm2.2; Recuperado 11 mayo del 2013 de: http://www.networkactiv.com/PIAFCTM.html

GLOSARIO DE TÉRMINOS

A

ARPA: Avanced Research Project Agency (Agencia de Investigación de Proyectos Avanzados)

C

Cabecera.- Información de control al comienzo de un mensaje, segmento, fragmento, paquete o bloque de datos.

Conector.- En inglés: 'socket'. Dirección que específicamente incluye un identificador de puerto, es decir, la concatenación de una dirección de internet con un puerto de TCP.

Conexión.- Un camino lógico de comunicación identificado por un par de conectores.

D

Datagrama.- Un mensaje que se envía en una red de comunicaciones de ordenadores por intercambio de paquetes.

Dirección de destino.- La dirección de destino, habitualmente los identificadores de red y de 'host'.

Dirección de origen.- La dirección de origen, generalmente los identificadores de red y de 'host'.

E

ETCP: Es una de las partes de un compilador que transforma su entrada en un árbol de derivación.

H

HDLC: Control de enlace de datos

I

IP.- El protocolo de internet o 'Internet Protocol'.

ICMP: Protocolo de Mensajes de Control de Internet

N

NLSP: Protocolo de Servicios Enlace NetWare

O

OSI: Interconexión de Sistemas Abiertos

P

Paquete.- Un conjunto de datos con una cabecera que puede estar o no lógicamente completa.

Parsing: Es una de las partes de un compilador que transforma su entrada en un árbol de derivación.

Puerto.- La porción de un conector que especifica qué entrada lógica o canal de salida se asocian con los datos.

Proceso.- Un programa en ejecución. Un origen o destino de datos desde el punto de vista de TCP o cualquier otro protocolo de 'host' a 'host'.

R

RFC: Describen, especifican y asisten en la implementación, estandarización y discusión de la mayoría de las normas, los estándares, las tecnologías y los protocolos relacionados con Internet y las redes en general.

S

Segmento.- Una unidad lógica de datos transferidos entre dos puntos.

SPX: Intercambio de Paquetes Secuenciados.

T

TCP: Protocolo de Control de Transmisión.

TLSP: Protocolo de Seguridad de la Capa de transporte

Anexo 1: Manual de usuario

VIRTUAL ROUTER

SIMULADOR PARA EL PROTOTIPO DE PROTOCOLO <<ETCP>>

MANUAL DE USUARIO

1. CONSIDERACIONES

El simulador ha sido construido bajo tecnología microsoft, por lo tanto si se requiere una funcionalidad al 100% de la aplicación se recomienda correrlo en sistemas operativos windows.

Se debe considerar que el desarrollo se hizo con .Net framework 4.0 por lo que la versión del sistema operativo debe ser al menos XP con service pack 3, de otro modo no se garantiza el funcionamiento correcto de la aplicación.

Al tratarse de un simulador que implementa un protocolo que no existe, es necesario realizar varias configuraciones adicionales a las requeridas en un protocolo existente.

La aplicación no implementa un modelo de programación multi-hilo por lo que; si no se sigue correctamente las instrucciones en el orden establecido es posible que el sistema se cuelgue o se cierre de manera espontánea.

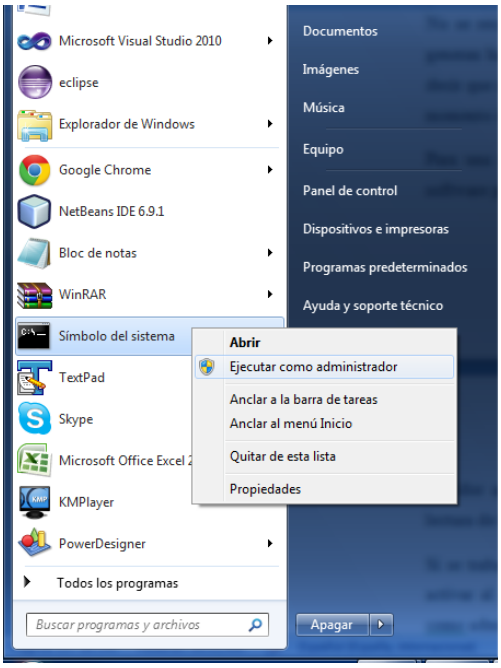
La aplicación utiliza implementaciones TCP para transportar los segmentos entre redes por lo tanto las mediciones sobre el funcionamiento en la red, serán las mismas mediciones que se realizara sobre el protocolo TCP.

No se recomienda crear redes clase B o C ya que la cantidad de hosts disponibles que se generan harán que sea lento el proceso de cada uno de ellos, no quiere decir que no se pueda pero NO SE RECOMIENDA; el hacerlo puede causar bloqueos al momento de la actualización automática e inestabilidad en el sistema.

Para una mejor visualización de los datos se recomienda levantar una instancia de la aplicación por cada red, ya que si no en la interfaz de monitoreo del servidor aparecerá toda la información de todas las redes, dificultando la lectura de la misma.

Si se trabaja con sistemas operativos windows vista o superiores es necesario crear un activar al súper usuario del sistema operativo para esto se necesita abrir la consola de windows como administrador

Figura 1: Consola como administrador



Elaborado por: Jorge Chapaca y David Rojas

Luego escribir el siguiente comando

Figura 2: Comando de activación del súper usuario

Elaborado por: Jorge Chapaca y David Rojas

Finalmente cerrar la sesión actual y volver a ingresar en la nueva cuenta creada.

2. INSTALACIÓN

Insertar el disco en la unidad lectora y ejecutar el archivo setup.exe, una vez ejecutado el instalador procederá a pre validar los requisitos previos, que en este caso en particular es el .Net framework 4.0 (solo en caso de que la computadora no lo posea).

Una vez ejecutado el archivo setup.exe seguir las siguientes instrucciones:

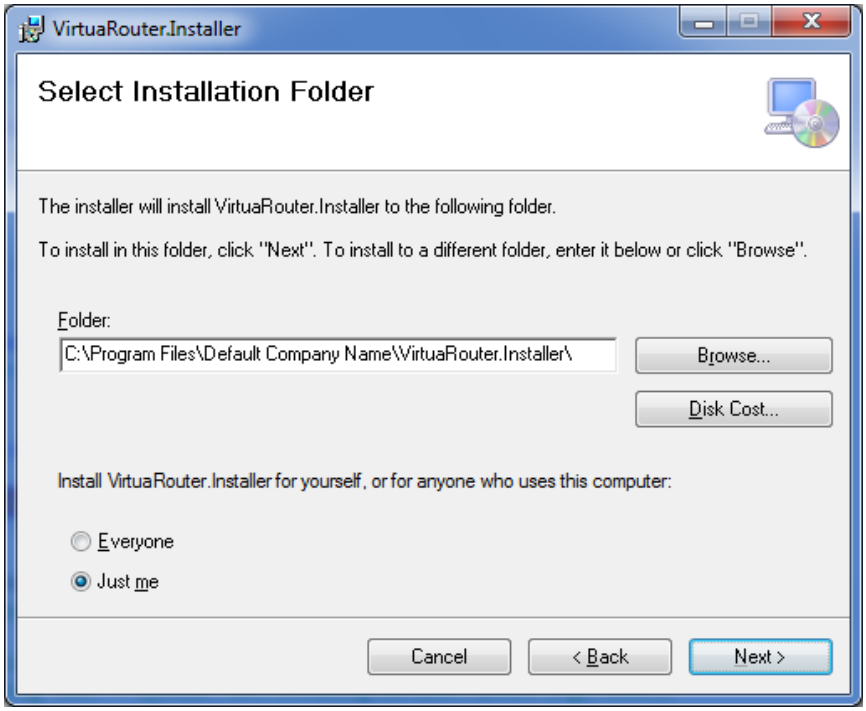
En la primera pantalla dar clic en next.

Figura 3: Pantalla uno de instalación

Elaborado por: Jorge Chapaca y David Rojas

Seleccionar el destino y se recomienda que la instalación sea para todos en el equipo

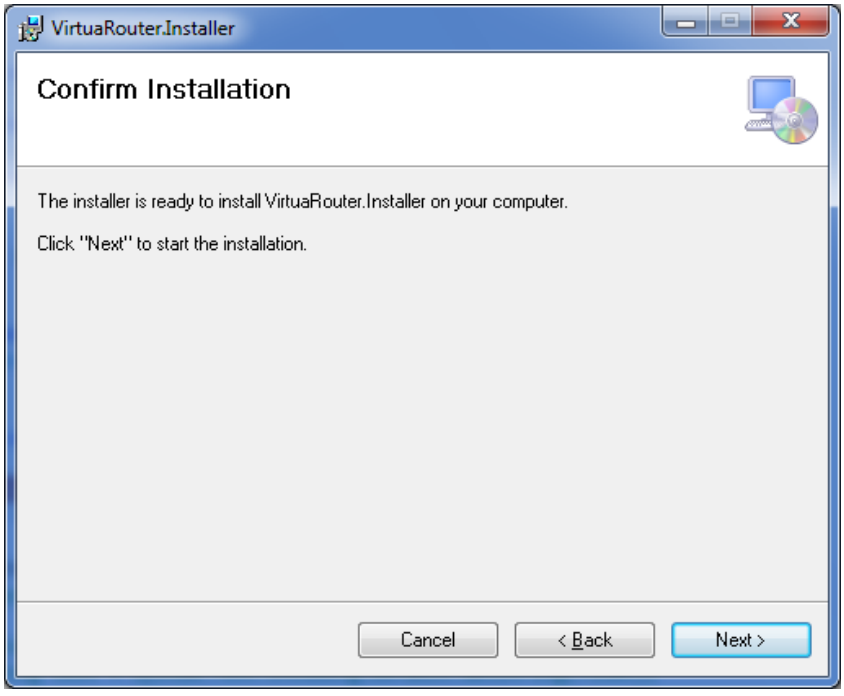
Figura 4: Pantalla dos de instalación



Elaborado por: Jorge Chapaca y David Rojas

Confirmar la instalación cuando esta haya terminado.

Figura 5: Pantalla final de instalación

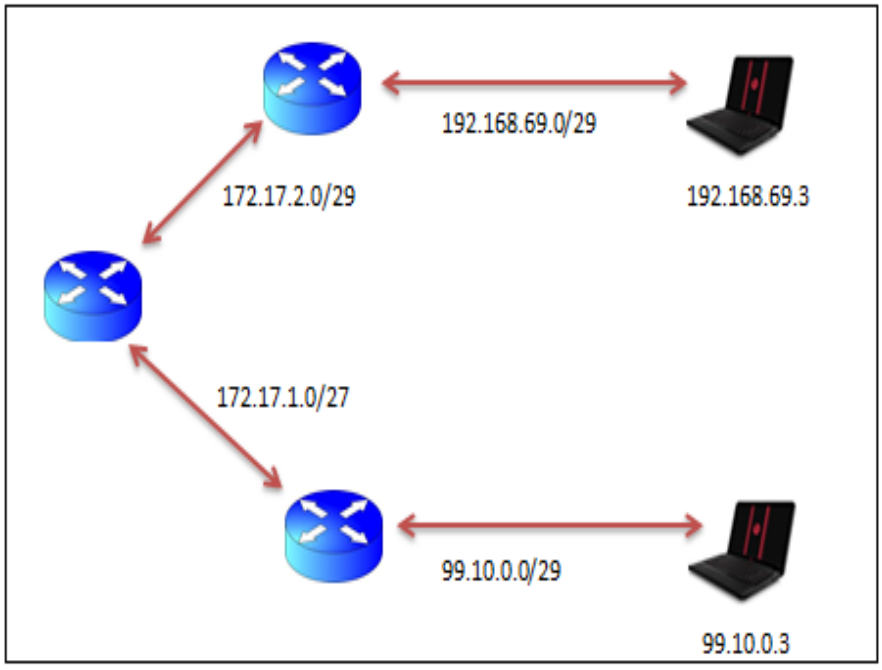


Elaborado por: Jorge Chapaca y David Rojas

3. PUESTA EN MARCHA

2.1 Escenario para la prueba

Figura 6: Escenario para pruebas

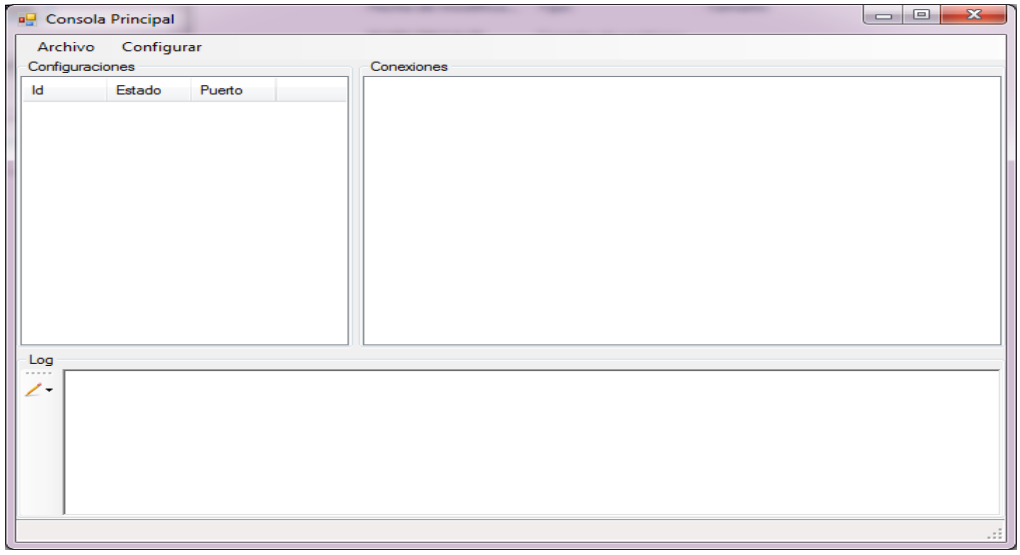


Elaborado por: Jorge Chapaca y David Rojas

2.2 Server

Ejecutar el acceso directo "Server Launcher" que se creó en el escritorio del computador, inmediatamente se obtendrá la interfaz del router básico.

Figura 7: Interfaz principal del servidor

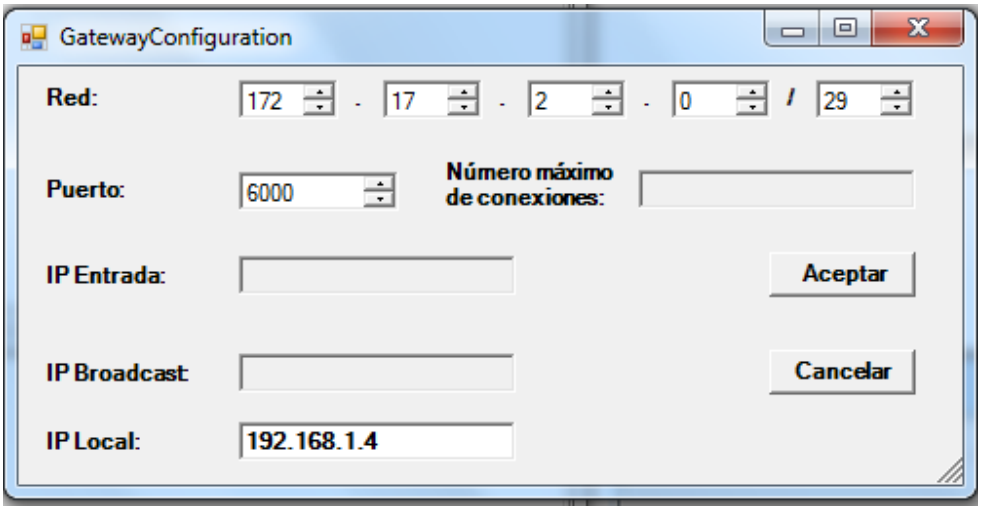


Elaborado por: Jorge Chapaca y David Rojas

La interfaz tiene de tres regiones, una región superior izquierda donde están las configuraciones de red, una región superior derecha donde están todos los host de la configuración, y una región inferior donde están las acciones que se tomen durante la ejecución del simulador.

Para crear una nueva configuración hay que seleccionar del menú superior la opción configurar → gateway.

Figura 8: Configuración de la red



Elaborado por: Jorge Chapaca y David Rojas

En el cuadro que se despliega ingresar la información correspondiente a la red que se pretende simular, siendo editables y obligatorios los campos de red, puerto e IP local.

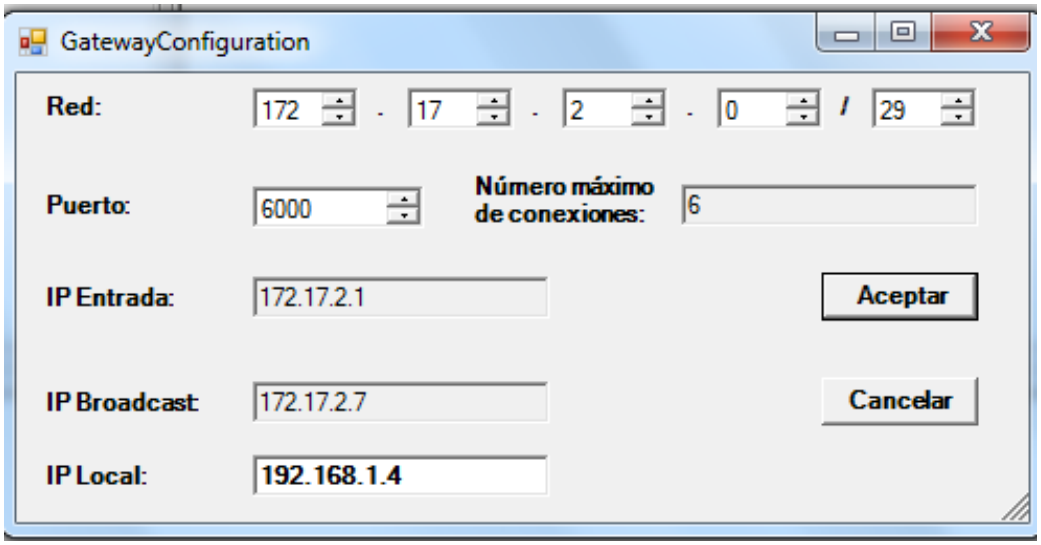
En red: Se debe ingresar la IP de red seguida de la submáscara, hay que señalar que al tratarse de una primera versión del producto, la aplicación recomendara usar submáscaras entre 25 y 30 para no afectar los recursos de la máquina y lograr verificar la funcionalidad mediante los logs en tiempo real.

El puerto: Es el punto de entrada y salida de las comunicaciones.

IP local: Es la dirección con la que está configurada la máquina física donde se levanta el servidor.

El resto de campos son llenados al dar clic en aceptar por primera vez.

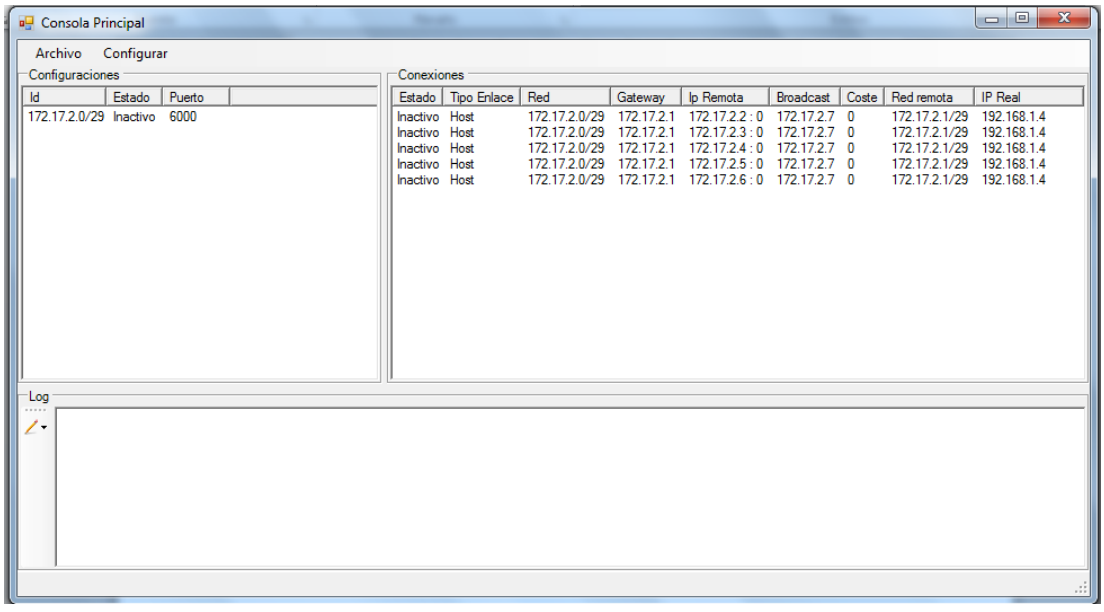
Figura 9: Auto completado de la configuración de la red



Elaborado por: Jorge Chapaca y David Rojas

Para que el sistema adapte la nueva configuración es necesario volver a dar clic en el botón aceptar, de esta manera se cargará la configuración a la interfaz principal.

Figura 10: Adición de una configuración de red

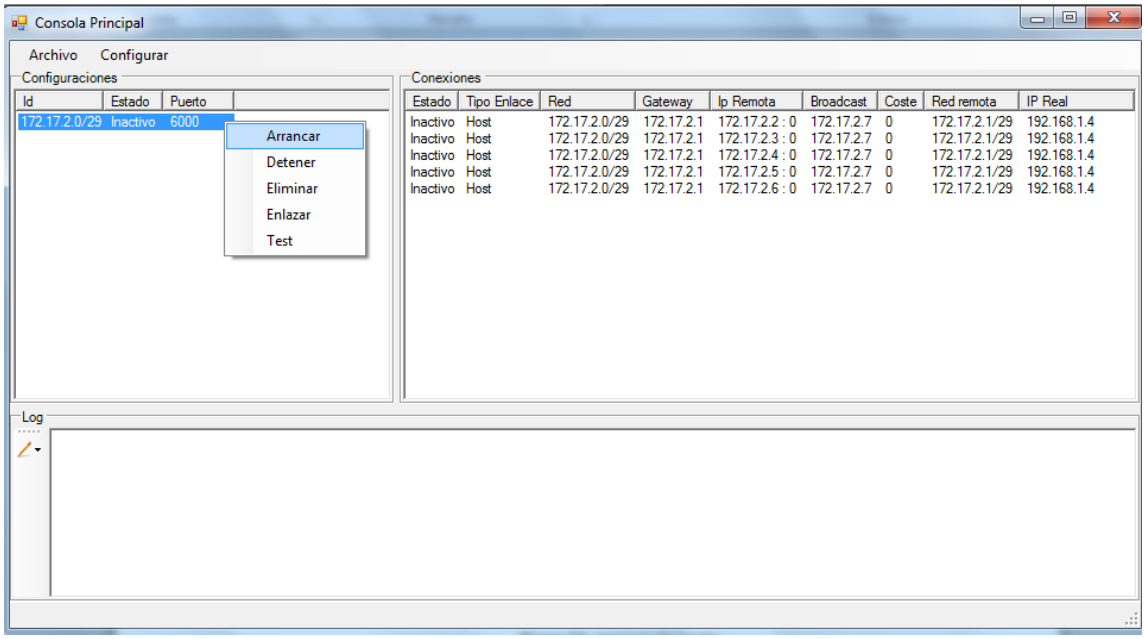


Elaborado por: Jorge Chapaca y David Rojas

Como se aprecia en la figura 10, en la parte izquierda se ha cargado la configuración creada, mientras que en la parte derecha están todos los hosts que pueden desprenderse de la red creada.

Para arrancar el servidor de una determinada red, lo que se debe hacer es dar clic derecho sobre la configuración deseada y seleccionar la opción Arrancar.

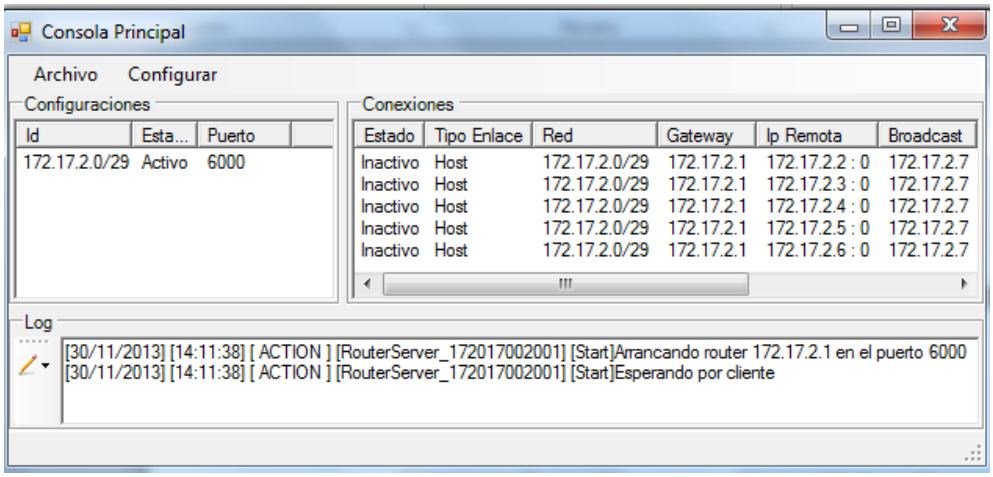
Figura 11: Arrancar el servidor



Elaborado por: Jorge Chapaca y David Rojas

Una vez que el servidor arranca, se puede apreciar que la región inferior se comienza a llenar automáticamente.

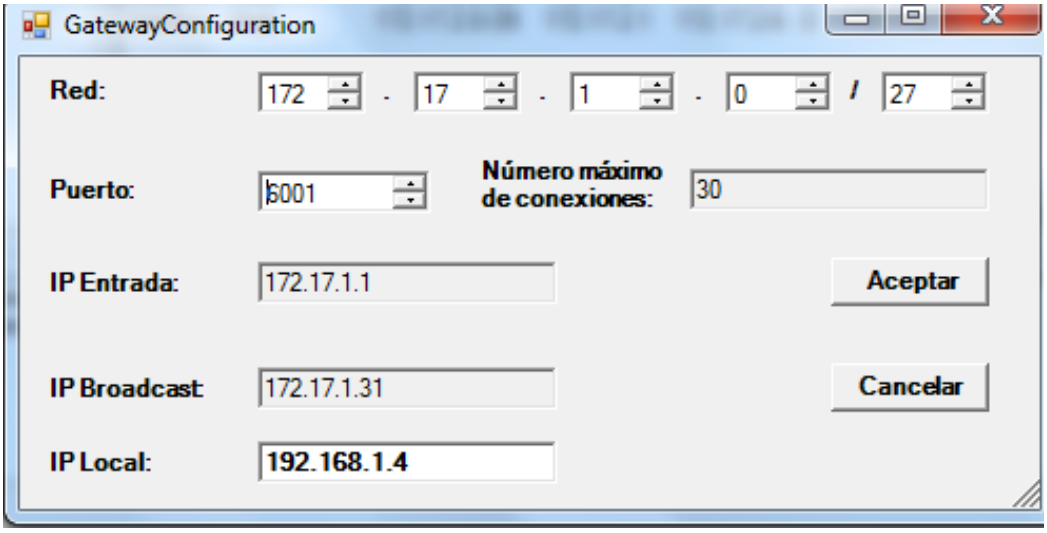
Figura 12: Log del servidor



Elaborado por: Jorge Chapaca y David Rojas

Dentro de la misma interfaz se pueden agregar diversas configuraciones, esto sirve para evitar abrir muchas veces el aplicativo, para lo cual se debe seguir el mismo procedimiento antes indicado.

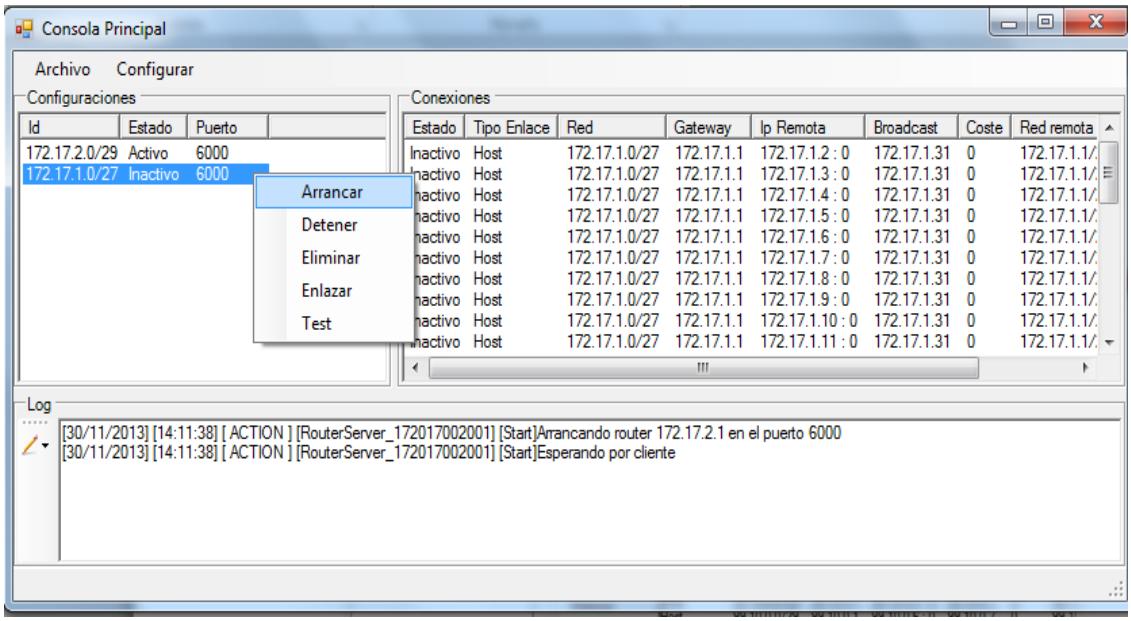
Figura 13: Segunda configuración



Elaborado por: Jorge Chapaca y David Rojas

Una vez aceptada la configuración se procederá a arrancar el servidor.

Figura 14: Arranque de segunda configuración

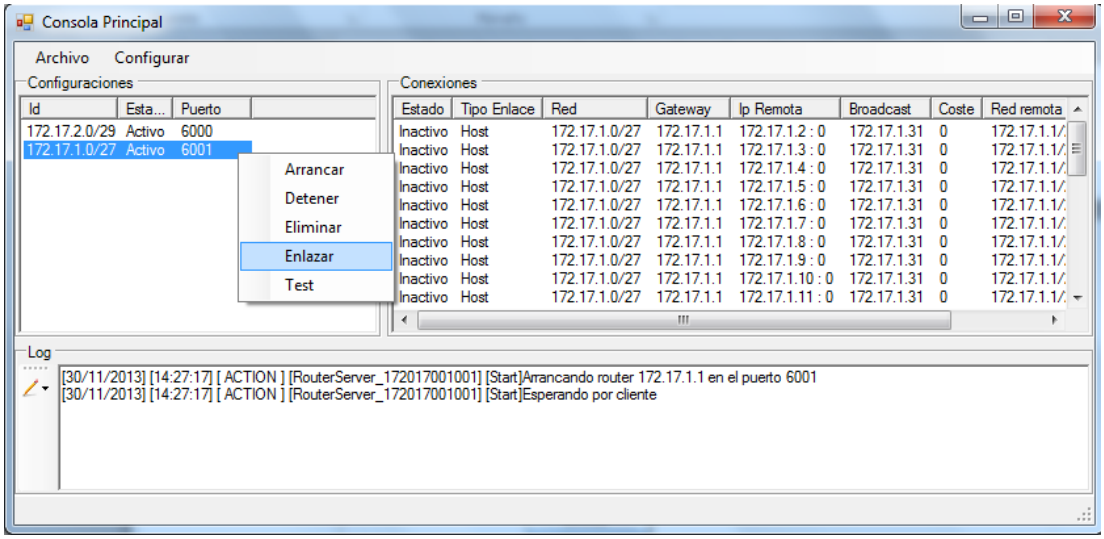


Elaborado por: Jorge Chapaca y David Rojas

Nota: el único cuidado que se debe tener en este punto es el de levantar configuraciones en puertos diferentes, ya que los protocolos no permiten más de un escucha en cada puerto.

Para enlazar dos redes lo que se debe hacer es seleccionar la configuración a la que desea agregar el enlace, dando clic derecho y luego seleccionando la opción enlazar, para el ejemplo se procederá a enlazar la red 172.17.1.0/27 hacia la 172.17.2.0/29, cabe señalar que solo será un enlace de ida, mas no de regreso; en caso de requerir el regreso se debe realizar el procedimiento que se detalla a continuación, pero desde la otra configuración.

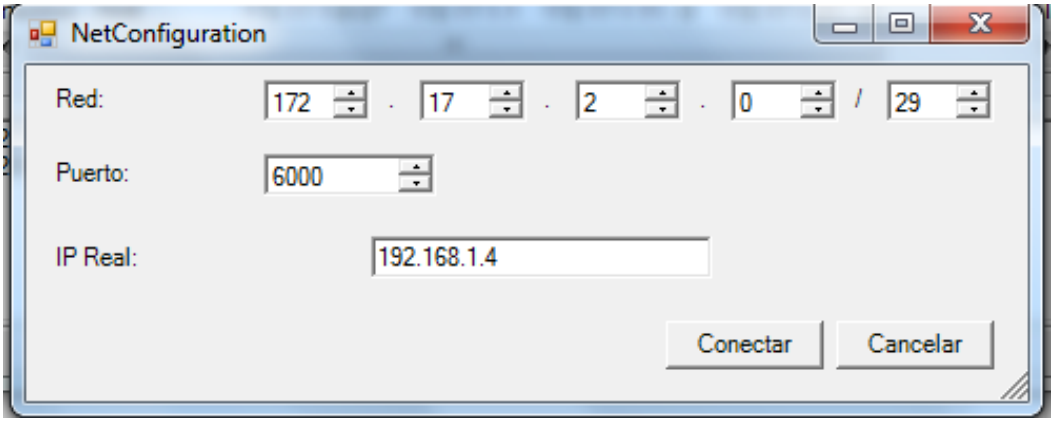
Figura 15: Enlazar redes



Elaborado por: Jorge Chapaca y David Rojas

En el cuadro de dialogo se deben ingresar los datos básicos para el enlace, se debe tener precaución de que el puerto que se escriba sea el mismo con el que se levantó la configuración destino, adicional se debe ingresar la IP real de conexión es decir la IP de la computadora donde se está ejecutando el software.

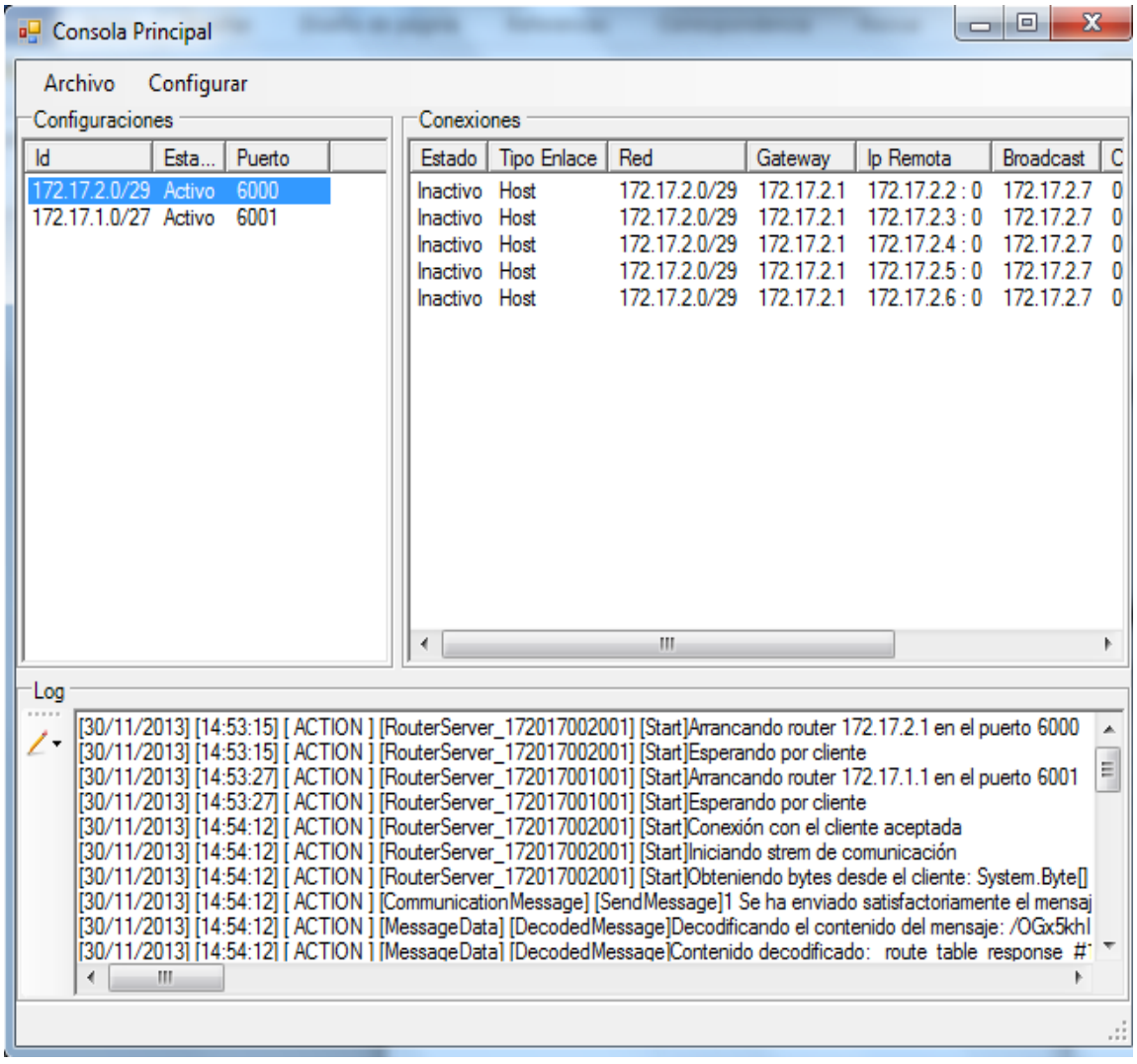
Figura 16: Enlazar redes, cuadro diálogo.



Elaborado por: Jorge Chapaca y David Rojas

Finalmente se puede apreciar los hosts disponibles.

Figura 17: Nueva conexión remota

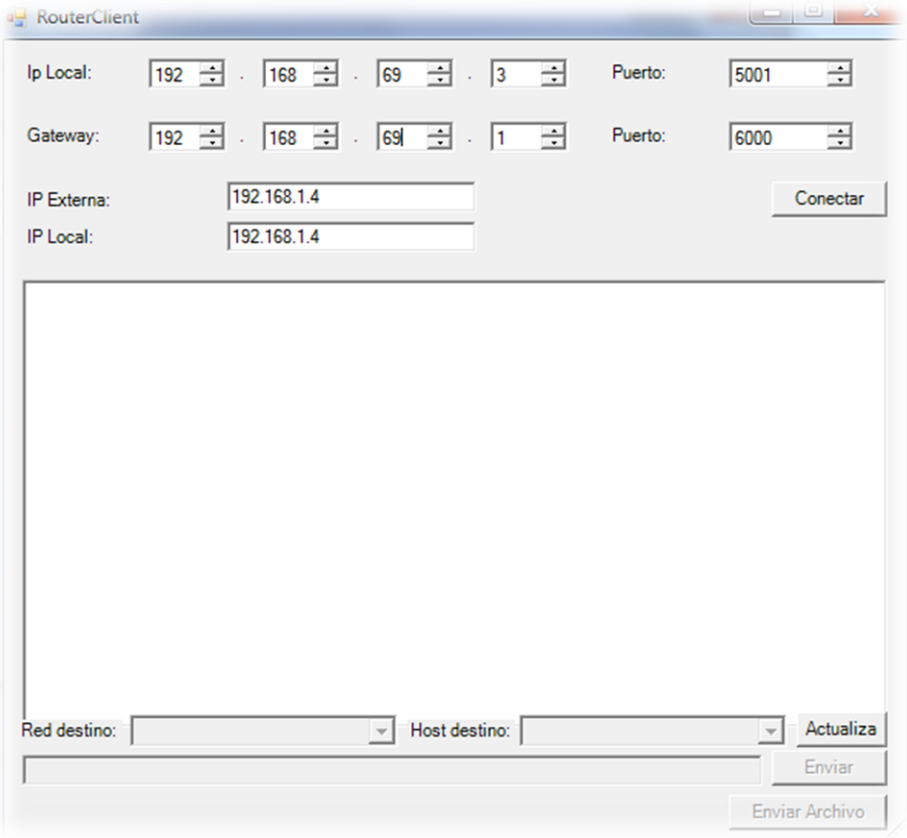


Elaborado por: Jorge Chapaca y David Rojas

2.3 Cliente

Para levantar el host cliente se debe ejecutar el archivo client.bat.

Figura 18: Interfaz cliente



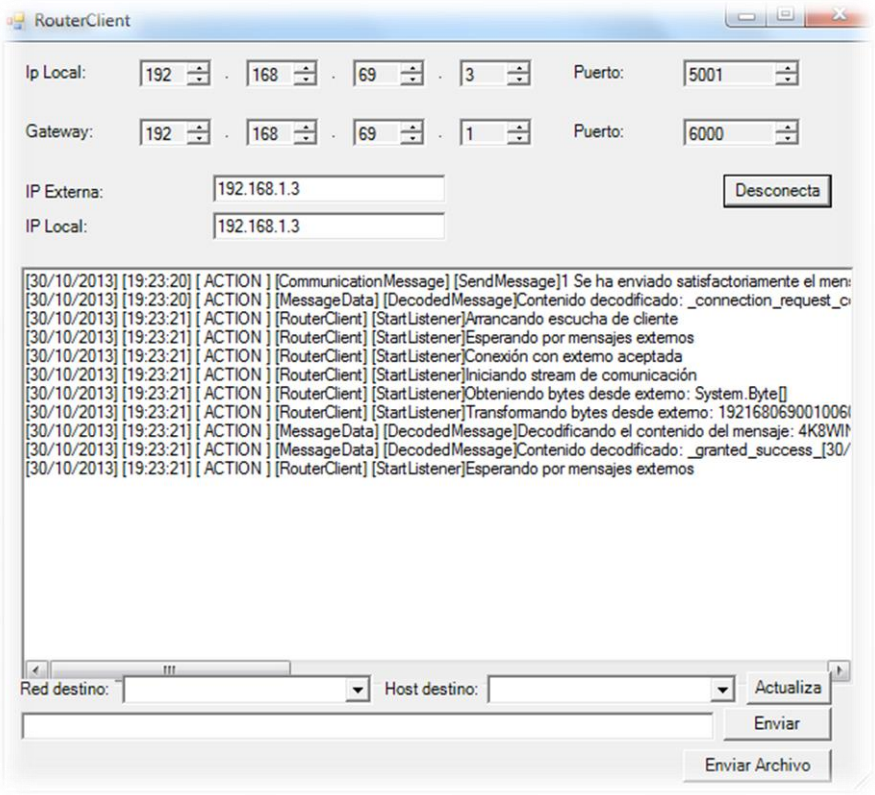
Elaborado por: Jorge Chapaca y David Rojas

De forma similar que el servidor, en el cliente se deben ingresar los datos de la configuración a la que se desea conectar siendo muy importante el puerto, IP local y computadora externa.

Para la dirección local se debe ingresar la IP con la que se va a conectar el cliente al simulador, más un puerto que servirá de gateway y este será la puerta de enlace de la red de destino a la que se requiere conectar, la computadora externa será la dirección real donde se está levantando el servidor y finalmente la dirección local es donde se está levantando el cliente.

Una vez ingresadas las configuraciones se da clic en el botón aceptar.

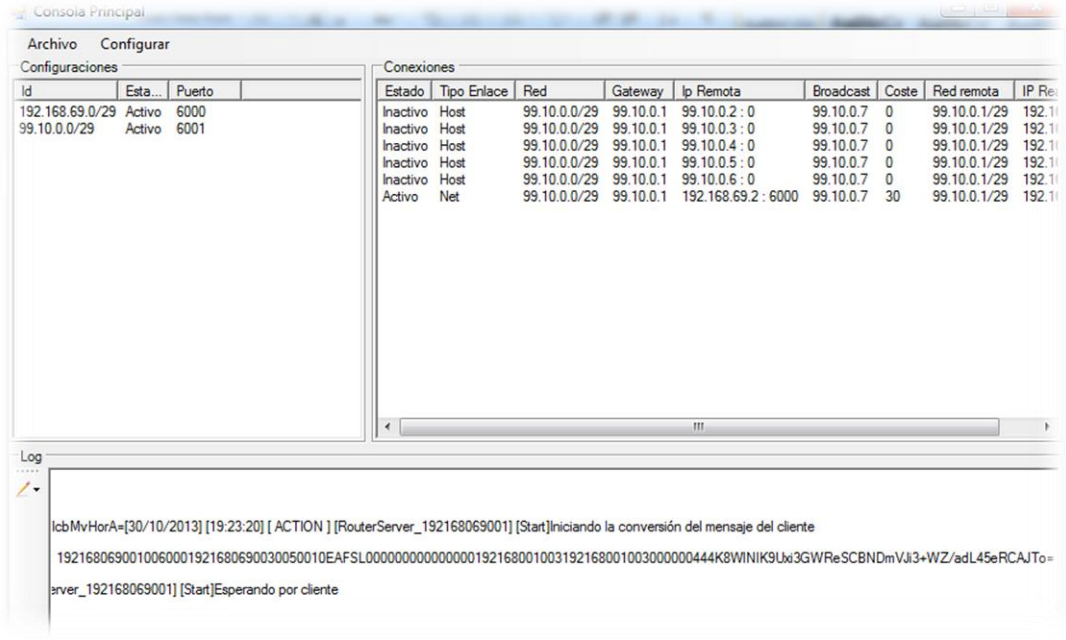
Figura 19: Conexión al servidor



Elaborado por: Jorge Chapaca y David Rojas

Mientras en el servidor se puede apreciar que se activa la terminal conectada

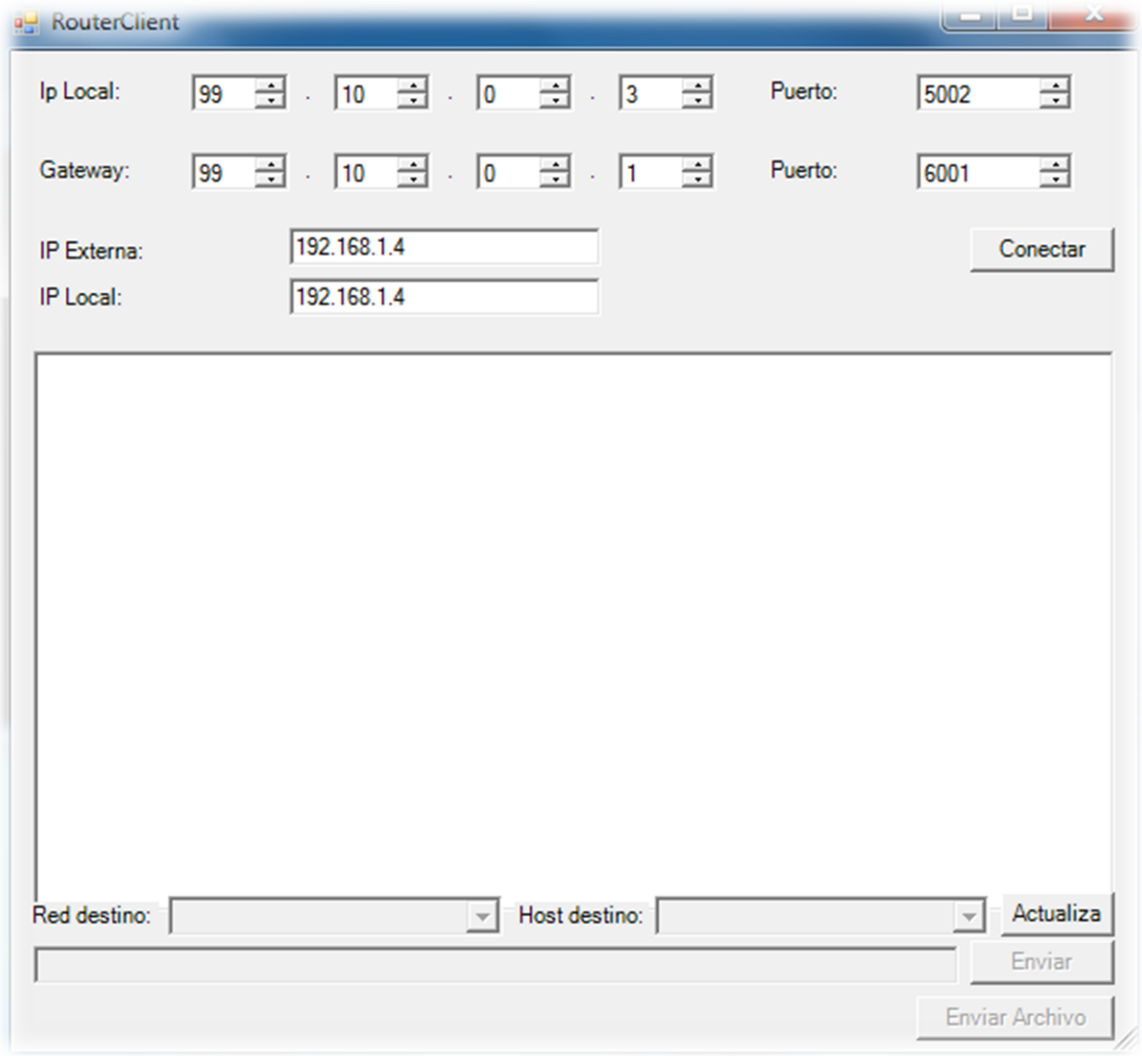
Figura 20: Servidor con terminal activa



Elaborado por: Jorge Chapaca y David Rojas

Con los mismos pasos se agrega un segundo host y quedara de la siguiente manera.

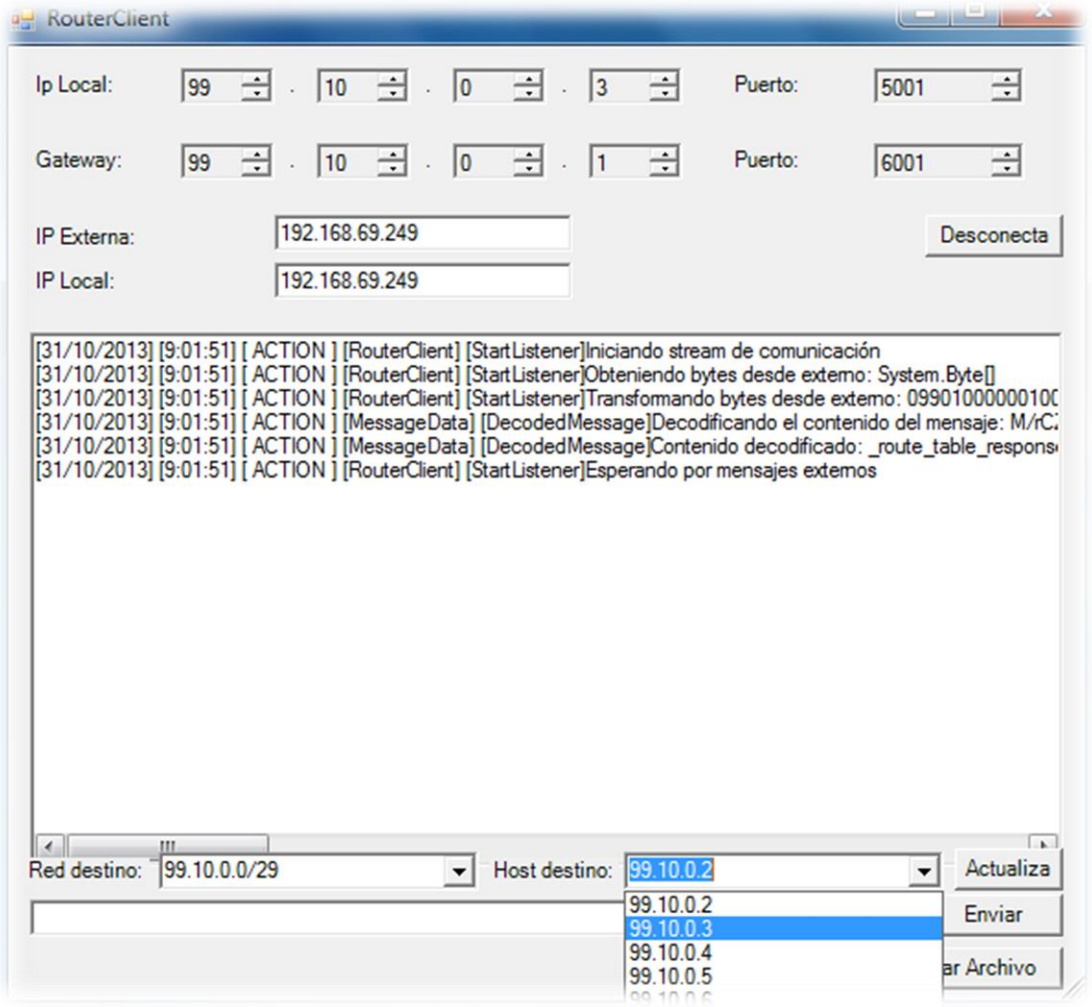
Figura 21: Hosts activos



Elaborado por: Jorge Chapaca y David Rojas

Finalmente para enviar datos de un host a otro en la parte inferior del cliente se tiene la información hacia los posibles destinos que puede tener el mensaje.

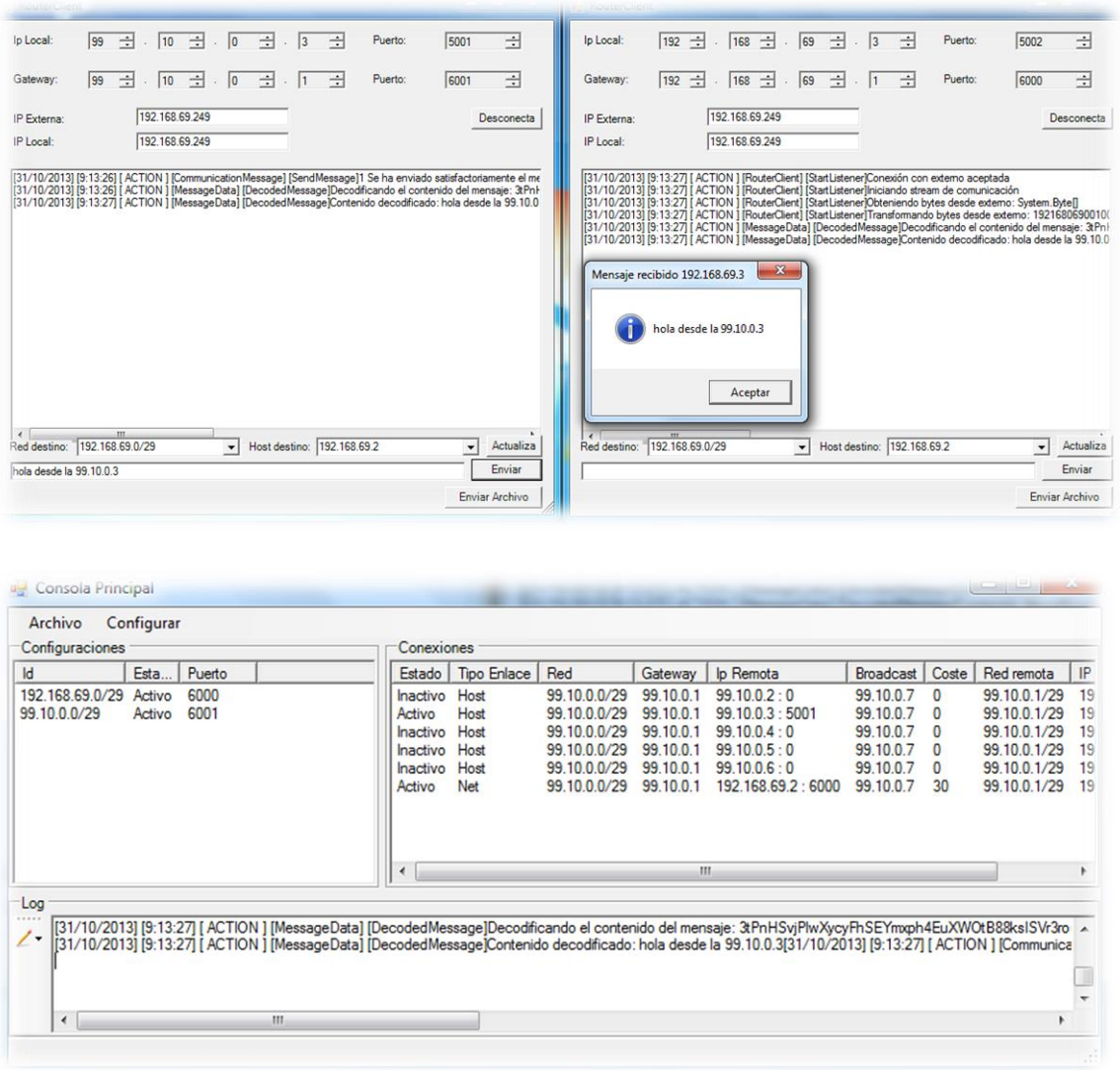
Figura 22: Posibles destinatarios



Elaborado por: Jorge Chapaca y David Rojas

Se envía un mensaje al segundo cliente conectado:

Figura 23: Interacción de componentes



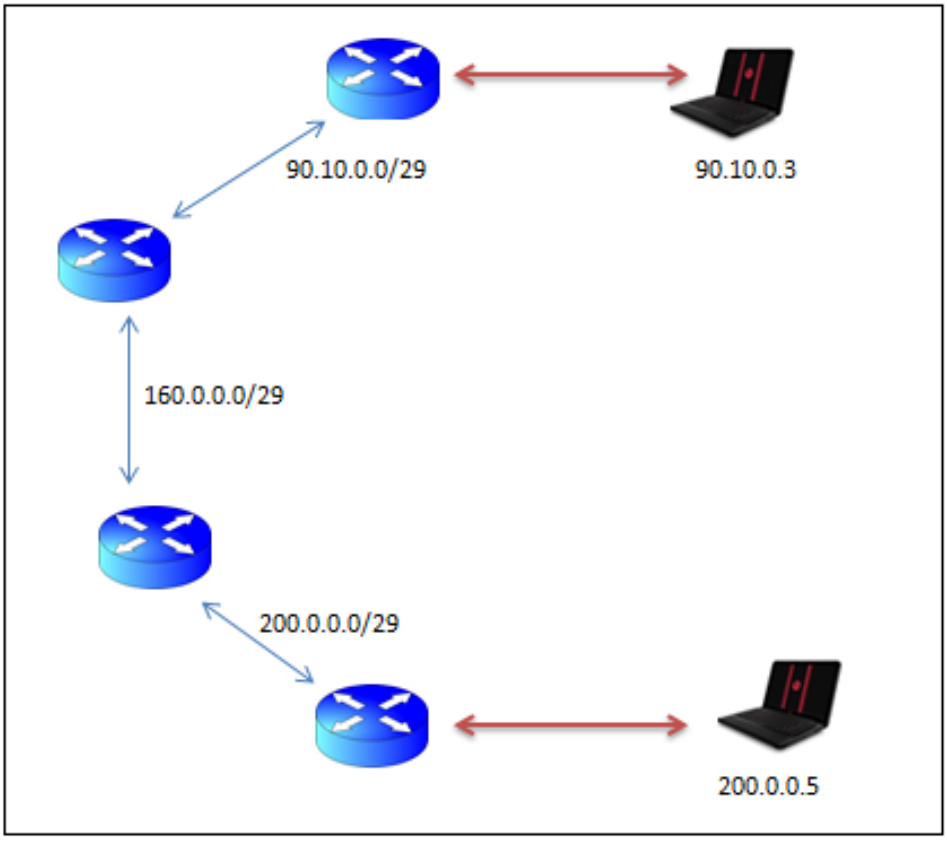
Elaborado por: Jorge Chapaca y David Rojas

En las figura se puede apreciar la interacción de los componentes, En logs de cliente y servidor que el mensaje pasó por cada uno de ellos.

EJEMPLO

Estructura: 3 routers y dos terminales en los extremos:

Figura 24: Caso de estudio 2

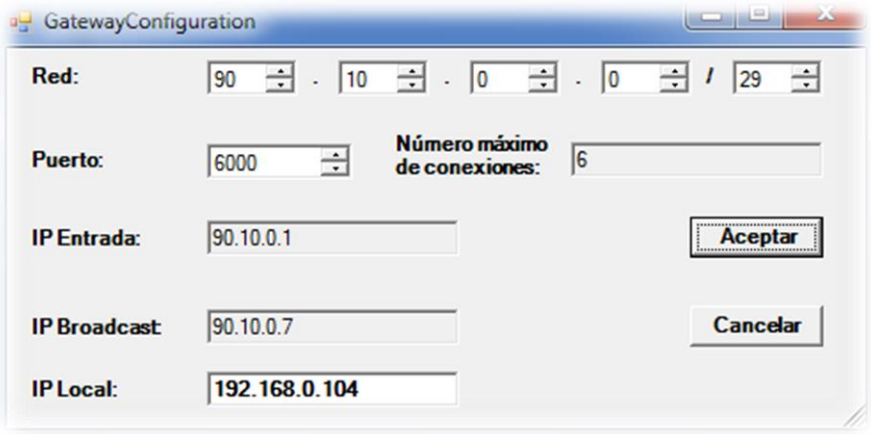


Elaborado por: Jorge Chapaca y David Rojas

Para el ejemplo se va a levantar 2 routers y un cliente en un equipo y lo restante en otro.

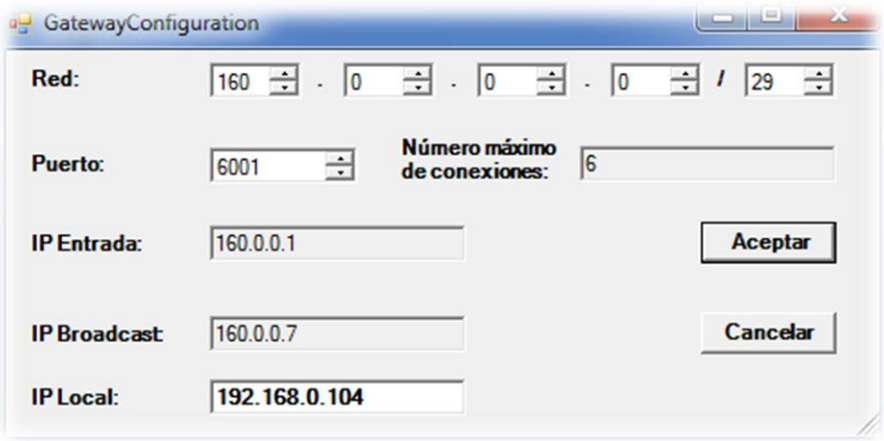
Con las indicaciones de los puntos anteriores se procede a levantar dos configuraciones de red en una de las máquinas, poniendo siempre la IP real que en la máquina uno es 192.168.0.4.

Figura 25: Configuración de red 1 en máquina 1



Elaborado por: Jorge Chapaca y David Rojas

Figura 26: Configuración de red 2 en máquina 1



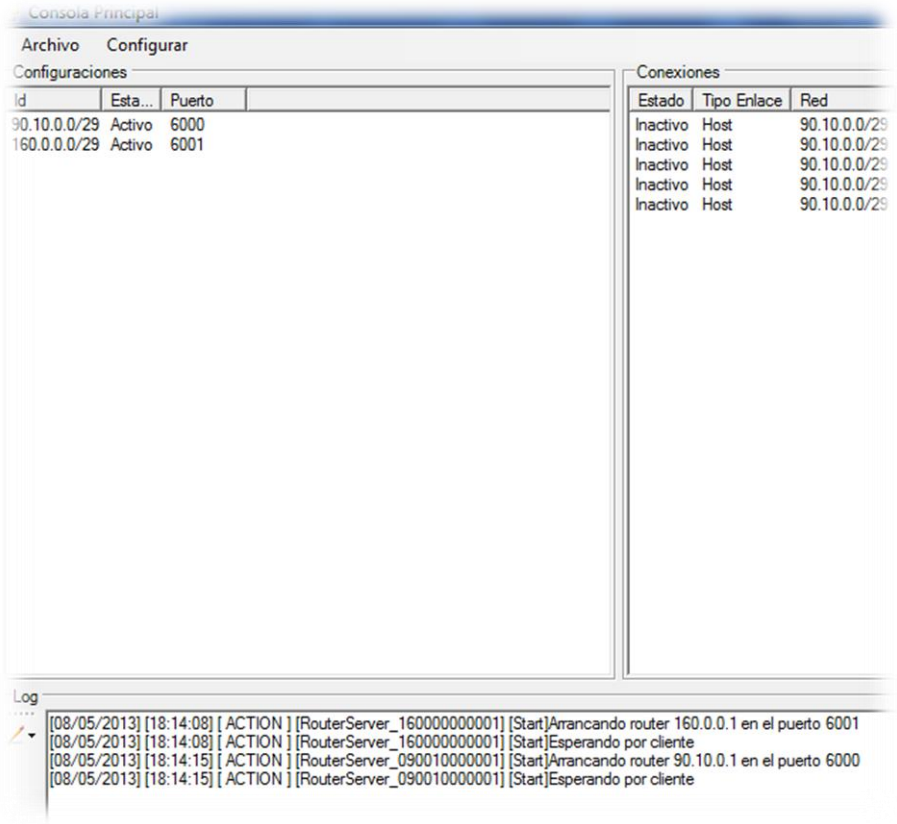
Elaborado por: Jorge Chapaca y David Rojas

Una vez levantadas las configuraciones es necesario arrancar las mismas:

Figura 27: Arrancando configuraciones

Elaborado por: Jorge Chapaca y David Rojas

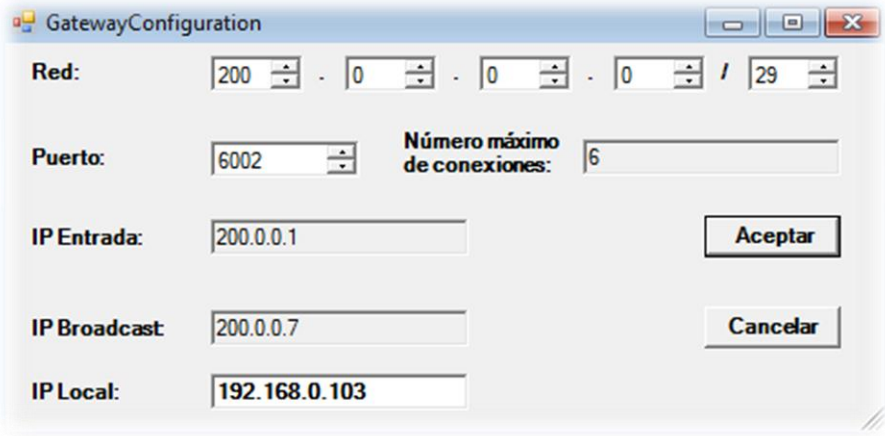
Figura 28: Configuraciones levantadas



Elaborado por: Jorge Chapaca y David Rojas

Luego se levanta una tercera configuración pero en una máquina aparte, para el ejemplo será la 192.168.0.103.

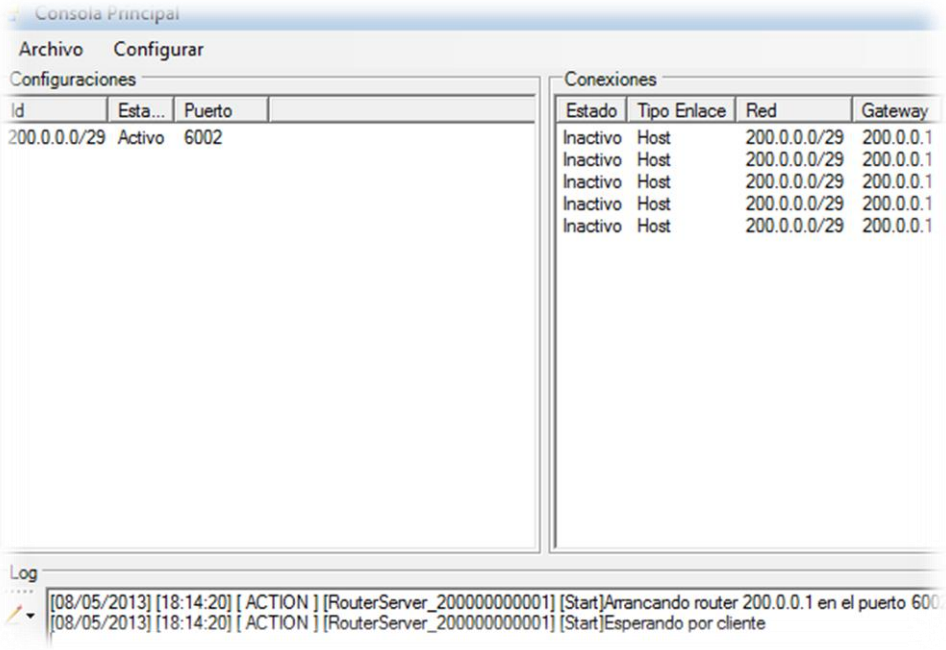
Figura 29: Configuración 3 máquina 2



Elaborado por: Jorge Chapaca y David Rojas

Y se levanta la misma

Figura 30: Configuración levantada de máquina 2

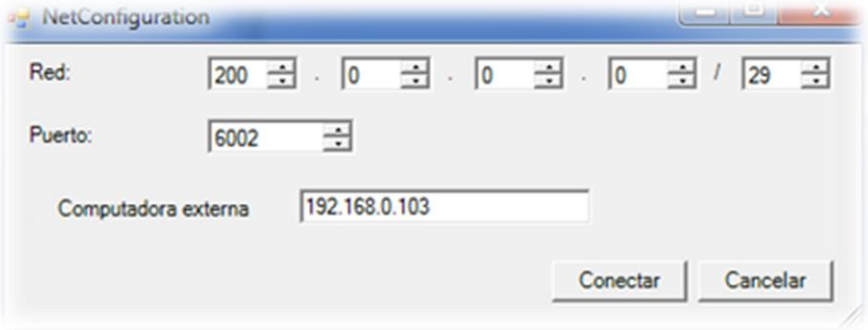


Elaborado por: Jorge Chapaca y David Rojas

La estructura que se quiere formar es 90.10.0.0/29 → 160.0.0.0/29 → 200.0.0.0/29, siendo configuraciones de la máquina uno 90.10.0.0/29 y 160.0.0.0/29, mientras 200.0.0.0/29 es de la máquina dos.

Una vez levantadas las configuraciones, es necesario enlazar las redes como se indicó previamente, entonces primero se crea el enlace 160.0.0.0/29 → 200.0.0.0/29.

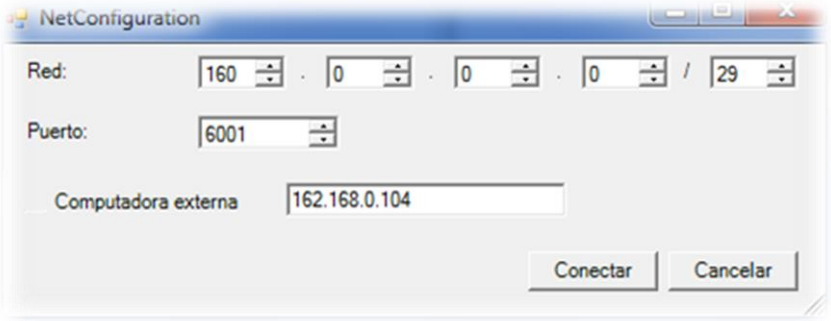
Figura 31: Enlace 160.0.0.0/29 hacia 200.0.0.0/29



Elaborado por: Jorge Chapaca y David Rojas

Luego el enlace 90.10.0.0/29 → 160.0.0.0/29

Figura 32: Enlace 90.10.0.0/29 a 160.0.0.0/29

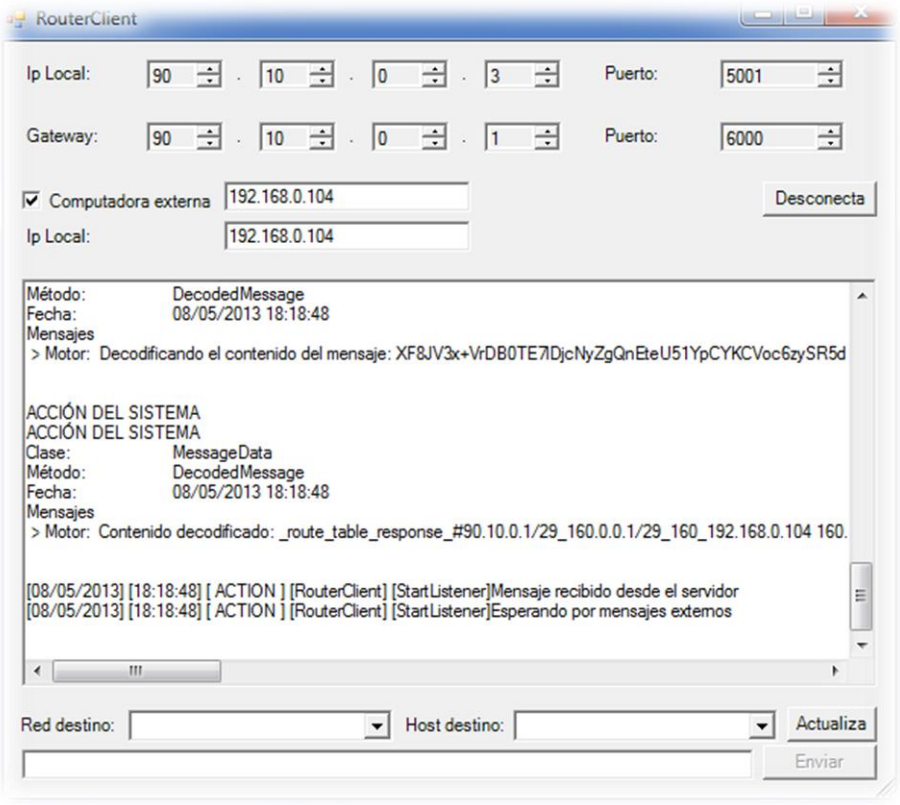


Elaborado por: Jorge Chapaca y David Rojas

Hay que señalar que estos enlaces se los debe levantar en el simulador de la máquina uno.

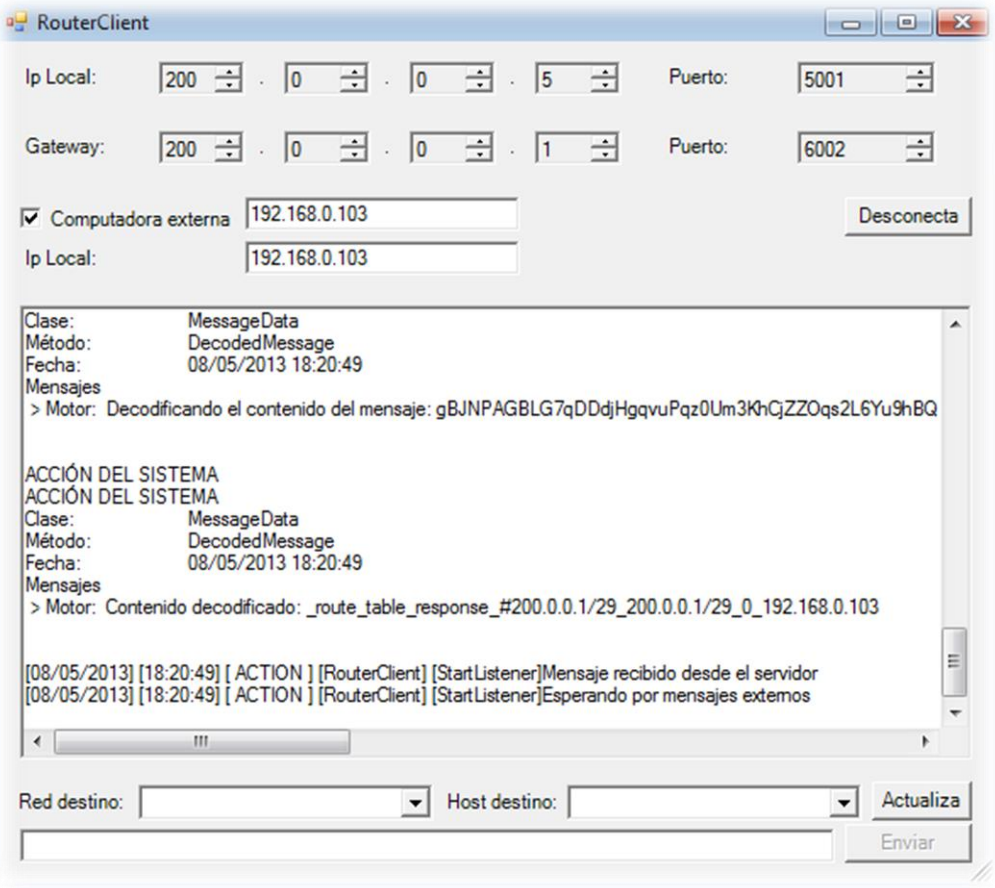
Como penúltimo paso la información se levantan los clientes, uno irá conectado hacia la red 90.10.0.0/29 y el otro a la red 200.0.0.0/29.

Figura 33: Cliente de red 90.10.0.0/29



Elaborado por: Jorge Chapaca y David Rojas

Figura 34: Cliente de red 200.0.0.0/29

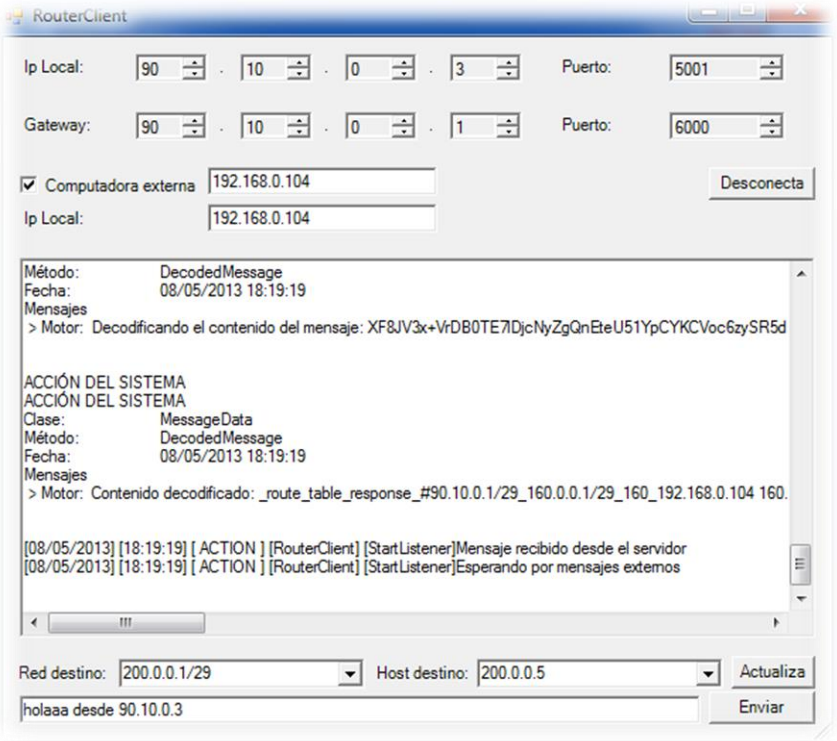


Elaborado por: Jorge Chapaca y David Rojas

Nota: Una vez conectados los clientes, temporalmente se debe dar clic una sola vez en el botón actualizar, esto es para que el cliente tome la información de la tabla de ruteo de los routers a los que se encuentran conectados.

Desde el cliente de la red 90.10.0.0/29 se envía un mensaje hacia el cliente de la red 200.0.0.0/29.

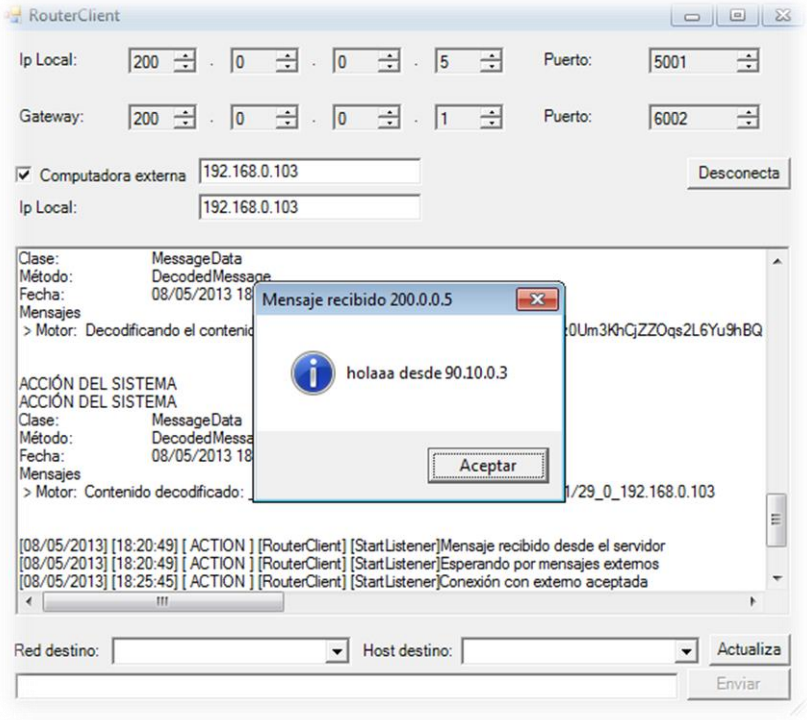
Figura 35: Mensaje desde el cliente de la red 90.10.0.0/29



Elaborado por: Jorge Chapaca y David Rojas

Y se observa cómo el mensaje llega hacia el cliente de la red 200.0.0.0/29

Figura 36: Mensaje obtenido en el cliente de la red 200.0.0.0/29

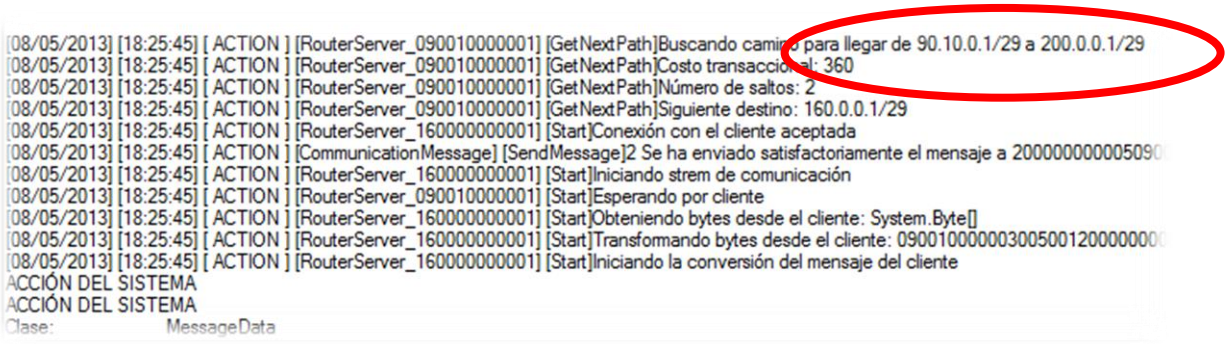


Elaborado por: Jorge Chapaca y David Rojas

Se puede apreciar en los logs de los servidores como va transportándose el mensaje entre redes, en la parte señalada se observará la búsqueda hacia el siguiente nodo y el lugar por donde se lo va a transportar.

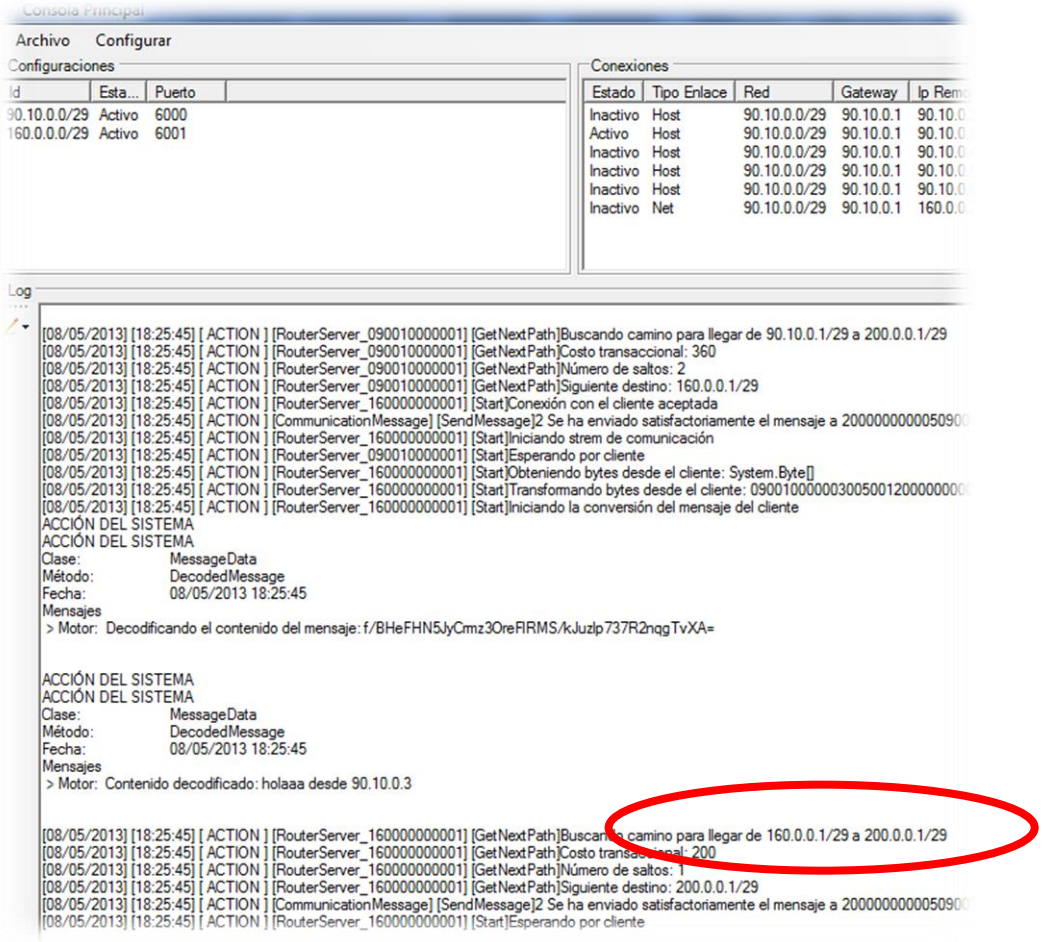
Figura 37: Paso de red 1 a red 2

Primero desde 90.10.0.0/29 → 160.0.0.0/29



Elaborado por: Jorge Chapaca y David Rojas

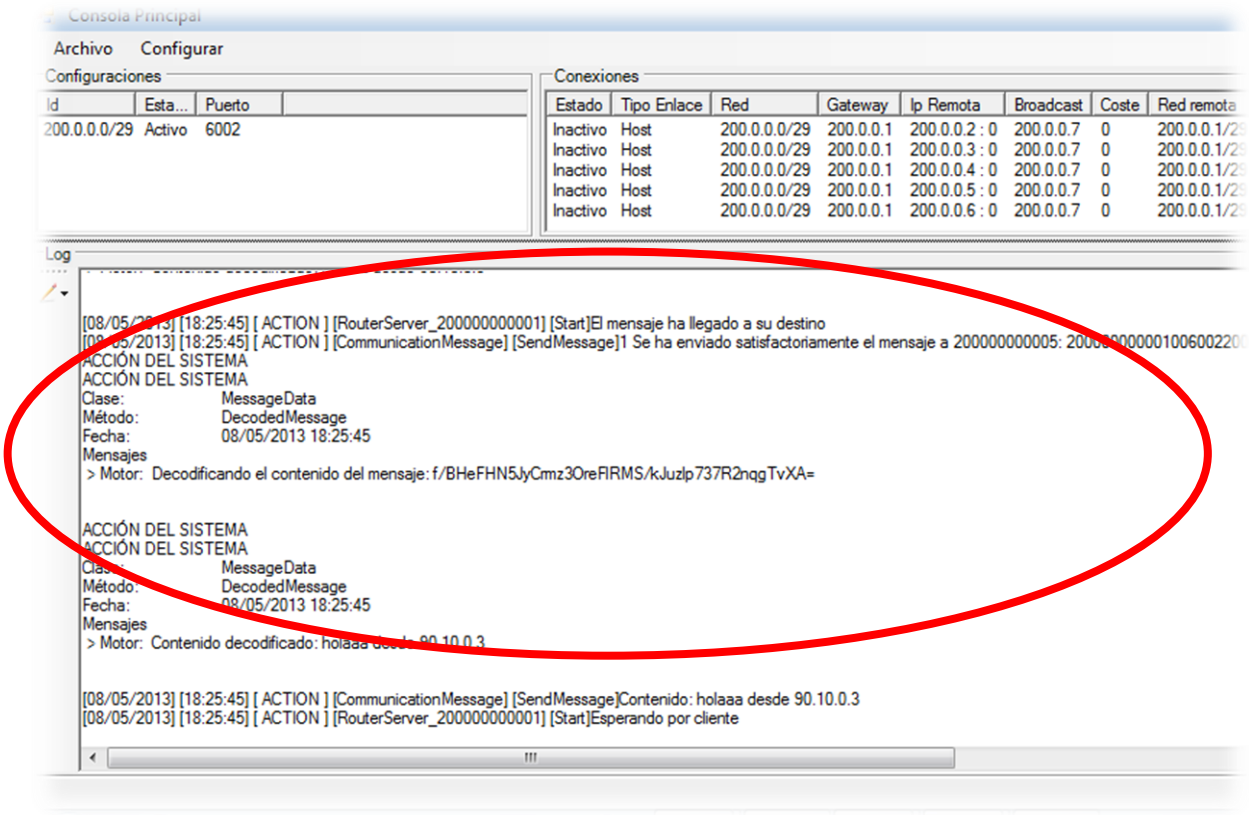
Figura 38: Paso de red 2 a red 3



Elaborado por: Jorge Chapaca y David Rojas

Finalmente el mensaje llega a la red 3 (200.0.0.0/29) para ser ruteado hacia el cliente

Figura 39: Llegada del mensaje a la red 3



Elaborado por: Jorge Chapaca y David Rojas

ARCHIVO router.properties

Dentro de la carpeta donde se instaló la aplicación, existe el archivo router.properties el cual es un archivo de propiedades compuesto por parámetros <clave>=<valor> los mismos que son leídos al momento de ejecutar una instancia de la aplicación, adicionalmente se pueden realizar comentarios anteponiendo el símbolo #.

Figura 40: Archivo router

Parámetro	Descripción	Ejemplo
active_delay	Bandera que indica si se activa o no el retraso tras el envío de cada parte del archivo	true / false

active_delay=true		
time_delay	Tiempo a esperar para enviar cada parte de un archivo, tiempo en milisegundos.	5000
time_delay=5000		
max_parts_file	Número de partes en las que se divide un archivo antes de ser enviado	20000
max_parts_file=20000		
local_ip	IP del equipo anfitrión	192.168.0.103
local_ip=192.168.0.103		
auto_update_server	Determina si el servidor se actualiza o no automáticamente	True
auto_update_server=false		
update_server_interval	Intervalo de tiempo en el que se actualiza el sevidor, en milisegundos	5000
update_server_interval=5000		
auto_update_client	Determina si el listener del cliente se actualiza o no automáticamente	False
auto_update_client=true		
update_client_interval	Intervalo en el que se actualiza el listener del cliente, en milisegundos.	5000
update_client_interval=15000		
auto_update_client_form	Determina si la interfaz del cliente se actualiza automáticamente o no	False

auto_update_client_form=true		
update_client_form_interval	Intervalo en el cual se actualiza la interfaz del cliente en milisegundos	5000
update_client_form_interval=15000		
pre_load_data	Determina si se van o no a pre cargar configuraciones al momento de iniciar una instancia de la aplicación.	True
pre_load_data=true		
number_nets	Indica el número de redes pre configuradas en la aplicación del servidor.	2
number_nets=1		
net_#	Donde # es el número de red pre configurada, por ejemplo si en number_nets está el valor 2, significa que tiene que existir dos registros: net_0 y net_1. Consta de tres partes separadas por una compa y sin espacios: ip, mascara de red, puerto.	90.10.10.0,29,6000
net_0=90.10.10.0,29,6000		
ip/subnet	Donde ip es la ip de red y subnet es la máscara. Son para precargar enlaces de una red, si por ejemplo en net_0 existe el valor 90.10.10.0,29,6000 en ip/subnet será el valor 90.10.10.0/29. Son 5 parámetros separados por una coma sin espacios:	200.10.15.30,29,6000,192.168.0.105,100

	Ip,máscara,puerto,ip real, coste.	
90.10.10.0/29=200.10.15.30,29,6000,192.168.0.105,100		

Elaborado por: Jorge Chapaca y David Rojas

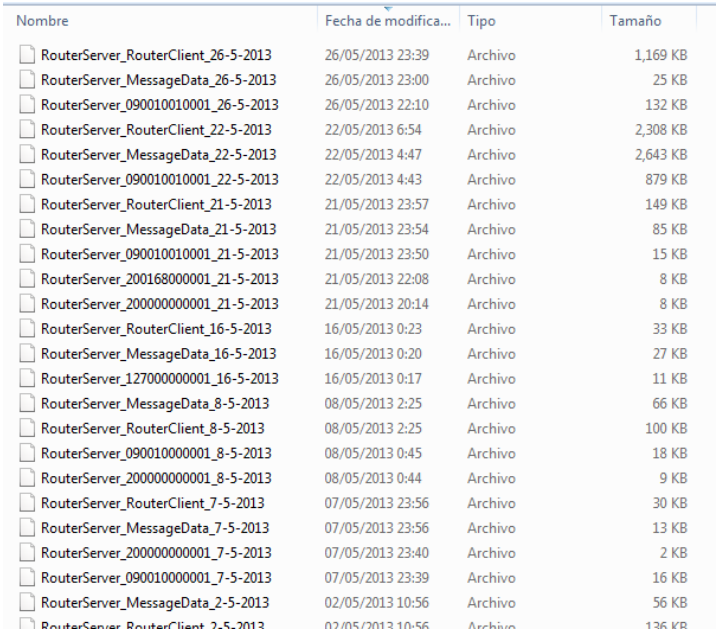
Contenido de ejemplo de un archivo router.properties:

```
max_parts_file=20000
local_ip=192.168.0.103
auto_update_server=false
update_server_interval=5000
auto_update_client=true
update_client_interval=15000
auto_update_client_form=true
update_client_form_interval=15000
#region para precargar las configuraciones
pre_load_data=true
number_nets=1
net_0=90.10.10.0,29,6000
#red_destino,mascara,puerto,ip_real,coste
90.10.10.0/29=200.10.15.30,29,6000,192.168.0.105,100
#190.150.0.0,29,6000,192.168.0.102,20
```

Log de la aplicación

La funcionalidad básica del log es la de registrar todos los eventos del sistema, con este fin se escribe en la pantalla de la aplicación y en archivos ubicados en el directorio de instalación dentro de una carpeta llamada serverlogs

Figura 41: Contenido de serverlogs



Nombre	Fecha de modifica...	Tipo	Tamaño
RouterServer_RouterClient_26-5-2013	26/05/2013 23:39	Archivo	1,169 KB
RouterServer_MessageData_26-5-2013	26/05/2013 23:00	Archivo	25 KB
RouterServer_090010010001_26-5-2013	26/05/2013 22:10	Archivo	132 KB
RouterServer_RouterClient_22-5-2013	22/05/2013 6:54	Archivo	2,308 KB
RouterServer_MessageData_22-5-2013	22/05/2013 4:47	Archivo	2,643 KB
RouterServer_090010010001_22-5-2013	22/05/2013 4:43	Archivo	879 KB
RouterServer_RouterClient_21-5-2013	21/05/2013 23:57	Archivo	149 KB
RouterServer_MessageData_21-5-2013	21/05/2013 23:54	Archivo	85 KB
RouterServer_090010010001_21-5-2013	21/05/2013 23:50	Archivo	15 KB
RouterServer_200168000001_21-5-2013	21/05/2013 22:08	Archivo	8 KB
RouterServer_200000000001_21-5-2013	21/05/2013 20:14	Archivo	8 KB
RouterServer_RouterClient_16-5-2013	16/05/2013 0:23	Archivo	33 KB
RouterServer_MessageData_16-5-2013	16/05/2013 0:20	Archivo	27 KB
RouterServer_127000000001_16-5-2013	16/05/2013 0:17	Archivo	11 KB
RouterServer_MessageData_8-5-2013	08/05/2013 2:25	Archivo	66 KB
RouterServer_RouterClient_8-5-2013	08/05/2013 2:25	Archivo	100 KB
RouterServer_090010000001_8-5-2013	08/05/2013 0:45	Archivo	18 KB
RouterServer_200000000001_8-5-2013	08/05/2013 0:44	Archivo	9 KB
RouterServer_RouterClient_7-5-2013	07/05/2013 23:56	Archivo	30 KB
RouterServer_MessageData_7-5-2013	07/05/2013 23:56	Archivo	13 KB
RouterServer_200000000001_7-5-2013	07/05/2013 23:40	Archivo	2 KB
RouterServer_090010000001_7-5-2013	07/05/2013 23:39	Archivo	16 KB
RouterServer_MessageData_2-5-2013	02/05/2013 10:56	Archivo	56 KB
RouterServer_RouterClient_2-5-2013	02/05/2013 10:56	Archivo	136 KB

Elaborado por: Jorge Chapaca y David Rojas

La figura 38 muestra el contenido de la carpeta serverlogs, los archivos s guardan con la siguiente modalidad:

RouterServer_<<ip>>_<<dia>>-<<mes>>-<<año>>

Se genera un archivo diario de log transaccional, por lo que se da mayor facilidad a la búsqueda de los mismos.

Anexo 2: Clase RouterServer.cs

La clase que implementa el algoritmo ETCP es RouterServer.cs, cuyo código se detalla a continuación:

- Constructor de la clase, el mismo que se encarga de inicializar todas las variables del sistema, recibe como parámetro un objeto NetGateway, el mismo que contiene la configuración de la red a levantar

```
public RouterServer(NetGateway localNetGateway)
{
    _localNetGateway = localNetGateway;
    _localIpAddress = localNetGateway.LocalConnection.Ip;
```

```
            _localPort = localNetGateway.LocalConnection.Port;
            _localKey = NetGateway.GetConnectorDataKey(_localIpAddress);
            _localLongPathKey =
Converter.ConvertIpAddressToString(_localIpAddress, true) + "/" +
localNetGateway.SubNetMask;

      // Diccionario de direcciones remotas
            _remoteConnections = new Dictionary<string, ConnectorData>();
      // Lista para nodos de enrutamiento
            _routerTablePath = new List<Path<string>>();
            _localPathKey = _localKey + "/" + _localNetGateway.SubNetMask;

      //Diccionario de costes (temporal)
            _tmpCost = new Dictionary<string, int>();
      // Tabla de enrutamiento
            _routerTable = new Dictionary<string, ConnectorData>();
      // Socket de comunicación
            _serverSocket = new TcpListener(IPAddress.Any, _localPort);
      //Bandera que indica si se ha arrancado el servicio
            _isLaunched = false;


//Timer de refrescamiento
            _serverTimer = new Timer();
            _serverTimer.Elapsed += RefreshServerData;
            _serverTimer.Interval =
Params.GetParam<int>("update_server_interval");
        }
```

- Método que arranca el hilo de ejecución del servidor

```
        public void LaunchServer()
        {
            _isLaunched = true; //Se indica que el servidor arranca
            _serverThread = new Thread(Start); //Se instancia el hilo del
servicio
            _serverThread.Start();//Se arranca la ejecución del hilo
        }
```

- Método de refrescamiento de datos, objetos de entrada son propios del delegado Elapsed del objeto Timer

```
    public void RefreshServerData(object source, ElapsedEventArgs e)
    {
    if (_isLaunched)
    {
    TestConnections(); //Si se ha levantado el servidor, se ejecuta el
método de prueba de conexión
    }
    }
```

- Método que detiene el servidor

```
       public void KillServer()
       {
       try
       { _serverSocket.Stop(); //Detiene el servidor de sockets
       }
       catch (Exception exception) //En caso de error escribe la causa en
el log
       {_logger.LogError(ClassName,  "KillServer",  "Error  al  tratar  de
detener el servicio");
        _logger.LogError(ClassName, exception);
        }
        if (_serverThread != null)
        _serverThread.Abort(); //Detiene la ejecución del hilo
        _isLaunched = false; //Cambia la bandera de estado de ejecución a
false
        _serverTimer.Enabled = false; //Detiene el timer}
```

- Método que arranca el servicio de sockets

```
        private void Start()
        {
        const string methodName = "Start"; //Identifica el método actual
        try
        {_serverSocket.Start(); //Arranca el servidor de sockets
         _isLaunched = true; //Cambia el estado de la bandera de ejecución
       //Establece si se activa o no el refrescamaniento automático
         _serverTimer.Enabled = Params.GParam<bool>("auto_update_server");
         }
       catch (SocketException exception) //Si existe un error se lo
escribe en la consola
        {_logger.LogError(ClassName, methodName, "Imposible arrancar el
servicio para " + NetGateway.GetConnectorDataKey(_localIpAddress) + " en
el puerto " + _localPort);
         _logger.LogError(ClassName, exception);
         _isLaunched = false;}
        _logger.LogStep(ClassName, methodName, "Arrancando router " +
NetGateway.GetConnectorDataKey(_localIpAddress) + " en el puerto " +
_localPort);

        while (_isLaunched)
         {_logger.LogStep(ClassName, methodName, "Esperando por cliente");
         TcpClient clientSocket = _serverSocket.AcceptTcpClient(); //Se
escuchan las conexiones entrants, si existe alguna se la acepta
         clientSocket.ReceiveTimeout = 10000; //Se establece timeout de
lectura de comunicaciones externas
         _logger.LogStep(ClassName, methodName, "Conexión con el cliente
aceptada");

           byte[] bytesFrom = new byte[10025]; //Arreglo que contrendrá la
cantidad de bytes a leer
          string dataFromClient = null; //Almacenará la data que ingresa

          _logger.LogStep(ClassName, methodName, "Iniciando strem de
comunicación");
```

```
           NetworkStream networkStream = clientSocket.GetStream(); //Se
obtiene el stream de red
           networkStream.Read(bytesFrom, 0,
(int)clientSocket.ReceiveBufferSize); //Se realize la lectura del stream
           _logger.LogStep(ClassName, methodName, "Obteniendo bytes desde
el cliente: " + bytesFrom);

           dataFromClient = Encoding.ASCII.GetString(bytesFrom); //Se hace
la transformación de bytes a string

           _logger.LogStep(ClassName, methodName, "Transformando bytes
desde el cliente: " + dataFromClient);

           _logger.LogStep(ClassName, methodName, "Iniciando la conversión
del mensaje del cliente");

           MessageData messageData = null; //Objeto que contiene la data
recibida
            try
            {
            messageData = new MessageData(dataFromClient); //Realiza el
parseo de la data recibida
            switch (messageData.ResponseMethod) //Se evalúa el tipo de
mensaje
            {case "PCC":
            {string id =
            NetGateway.GetConnectorDataKey(

MessageTemplates.ConvertToIpAddress(messageData.LocalIpAddress));

           //Primero verificar si la dirección a la que se va a conectar
es válida if
(messageData.RemoteIpAddress.Equals(Converter.ConvertIpAddressToString(_lo
calIpAddress, true)))
                                {
         //Si la dirección del solicitante es válida, entonces se envía un
mensaje de conexión aceptada.

CommunicationMessage.SendMessage(MessageTemplates.GrantedConnectionRequest
(messageData));
             //Se agrega al diccionario de conexiones remotas
            _remoteConnections[id].Port = messageData.LocalPort;
                _remoteConnections[id].Client = clientSocket;
                      _remoteConnections[id].RemoteIp =
      MessageTemplates.ConvertToIpAddress(messageData.RealSourceIp);
                                }
else //Si el acceso es hacia una ip distinta a la que se requiere entonces
                             negar la conexión
                                {
                    CommunicationMessage.SendMessage(
       MessageTemplates.RevokeGrantedConnectionRequest(messageData,
        "El gateway al que trata de conectarse no es correcto."));
                                }
                                }
                              break;
                            case "SSF":
                                {
             string[] data = messageData.NetOwner.Split('/');
       IPAddress net = MessageTemplates.ConvertToIpAddress(data[0]);
```

```
  string netKey = Converter.ConvertIpAddressToString(net, false) + "/" +
                                    data[1];

                                IPAddress to =
    MessageTemplates.ConvertToIpAddress(messageData.RemoteIpAddress);

if (_localPathKey.Equals(netKey)) //Si la Ip destino es igual a la Ip del
         servidor entonces es porque es el servidor destinatario
                                    {
    _logger.LogStep(ClassName, methodName, "El mensaje ha llegado a su
                                     destino");
                  //Se envía el mensaje al host destino
                SendFileMessage(messageData.MessageContent,
     _remoteConnections[Converter.ConvertIpAddressToString(to, false)
                               ], messageData);

                                    }
   Else //Caso contrario se realize el envío al lugar donde se requiera
                                    {
                            TcpClient tcpClient;
                  int maxResends = MaxResendMessage;
                            int senderCount = 0;
                  do //Se realize un intent obligatorio
                                    {
if (senderCount == maxResends) //Siempre y cuando el número de intentos no
                       exceda el máximo permitido (3)
                                     break;
                         tcpClient = new TcpClient();
string nextData = GetNextPath(netKey); //Se obtiene la ruta mas corta para
                                llegar al destino
                  string nextId = nextData.Split('/')[0];
                                     try
                                     {
  //Se establece un cliente de comunicación temporal para el envío de la
                                     data
          tcpClient.Connect(_remoteConnections[nextId].RemoteIp,
                     _remoteConnections[nextId].Port);
                                     }
                                    catch
                                     {
                                tcpClient = null;
                                     }
                                  senderCount++;
 } while (!CommunicationMessage.SendMessage(messageData, tcpClient)); //Se
                                envía los datos

                       if (senderCount == maxResends)
                { //Se ecribe el mensaje de error en el log
                  _logger.LogError(ClassName, methodName,
"Imposible entregar el mensaje a la red, no se puede hallar una ruta para
                                llegar a " +
                                  netKey);
                                     }

                                     }
                                     }
                                   break;
                                 case "EAF":
```

```
                                    {
     if (Constants.GrantedSuccess.Equals(messageData.MessageContent))
                                    {
//Si se recibe un mensaje de conexión exitosa se establece la comunicación
                                IPAddress from =
     MessageTemplates.ConvertToIpAddress(messageData.LocalIpAddress);
            string key = NetGateway.GetConnectorDataKey(from);

             _remoteConnections[key].Client = new TcpClient();

                                    }
 else if (messageData.MessageContent.StartsWith(Constants.AddPathRequest))
    { //Si es una petición para agregar a table de ruteo se lo agrega

                                    string[] data =
messageData.MessageContent.Split('#');
                                    string source = data[1];
                                    string destination = data[2];
                                    int cost = Convert.ToInt32(data[3]);

                                    AddPath(source, destination, cost);
//Se agrega a la lista de nodos de ruteo
                                }
                                else if
(messageData.MessageContent.StartsWith(Constants.RouteTableResponse))
                                { // Si se recibe una tabla de ruteo
complete desde otro servidor, entonces se agrega elemento por elemento a
la tabla actual

                                    string[] data =
messageData.MessageContent.Split('#');
                                    string[] nets = data[1].Split(' ');
                                    foreach (string net in nets)
                                    {
                                        string[] internalData =
net.Split('_');
                                        if (internalData.Length > 1)
                                        {
                                            string source =
internalData[0];
                                            string destination =
internalData[1];

                                            if
(!source.Equals(destination))
                                            {
                                                int cost =
Convert.ToInt32(internalData[2]);

                                                AddPath(source,
destination, cost);
                                            }
                                        }
                                    }
                                }
                                else
                                {
             //Se lo trata como un mensaje de comunicación
```

```
                                    string[] data =
messageData.NetOwner.Split('/');
                                    IPAddress net =
MessageTemplates.ConvertToIpAddress(data[0]);

                                    string netKey =
Converter.ConvertIpAddressToString(net, false) + "/" + data[1];

                                    IPAddress to =
MessageTemplates.ConvertToIpAddress(messageData.RemoteIpAddress);

                                    if (_localPathKey.Equals(netKey)) //Si
la ip destino es igual a la actual, entonces llego a su destino
                                    {
                                        _logger.LogStep(ClassName,
methodName, "El mensaje ha llegado a su destino");
                    //Se envía el mensaje al host destino

SendTextMessage(messageData.MessageContent,

_remoteConnections[Converter.ConvertIpAddressToString(to, false)
                                                                    ]);
                                    }
                                    else
                                    {
             //Caso contrario se busca la ruta mas corta para llegar al
destino
                                        TcpClient tcpClient;
                                        int maxResends = MaxResendMessage;
                                        int senderCount = 0;
                                        do
                                        {
                                            if (senderCount == maxResends)
                                                break;
                                            tcpClient = new TcpClient();
                                            string nextData =
GetNextPath(netKey);
                                            string nextId =
nextData.Split('/')[0];
                                            try
                                            {

tcpClient.Connect(_remoteConnections[nextId].RemoteIp,

_remoteConnections[nextId].Port);
                                            }
                                            catch
                                            {
                                                tcpClient = null;
                                            }
                                            senderCount++;
                                        } while
(!CommunicationMessage.SendMessage(messageData, tcpClient));

                                        if (senderCount == maxResends)
                                        {
                                            _logger.LogError(ClassName,
methodName,
```

```
                                                                    "Imposible
entregar el mensaje a la red, no se puede hallar una ruta para llegar a "
+
                                                                    netKey);
                                            }

                                        }
                                    }
                                }
                                break;
                            case "EAC":
                                {
                                    if
(Constants.RouteTableRequest.Equals(messageData.MessageContent))
                                    {
                                        SendRouteTable(messageData); //Se
envía la tabla de ruteo
                                    }
                                }
                                break;
                            default:
                                _logger.LogError(ClassName, methodName,
"Método de respuesta desconocido: " + messageData.ResponseMethod);
                                break;
                        }
                    }
                    catch (Exception exception)
                    {
                        _logger.LogError(ClassName, methodName, "Se ha
recibido un paquete de estructura desconocida: " + dataFromClient);
                        _logger.LogError(ClassName, exception);
                    }
                }
            }
```

- Método que arma la tabla de ruteo para que sea enviada

```
        private bool SendRouteTable(MessageData messageData)
        {
            string response = Constants.RouteTableResponse + "#";
      //Para cada element de la lista de nodos
            foreach (Path<string> path in _routerTablePath)
            {
      //Se obtiene la red
                string key = path.Destination.Split('/')[0];
                if (_remoteConnections.ContainsKey(key))
                    key =
NetGateway.GetConnectorDataKey(_remoteConnections[key].RemoteIp);
                else
                    key = _localKey;
      //Se concatena en un string toda la data a enviar
                response += path.Source + "_" + path.Destination + "_" +
path.Cost + "_" + key + " ";

            }
```

```
      //Se concatena la red actual también
            response += _localKey + "/" + _localNetGateway.SubNetMask +
"_" + _localKey + "/" +
                        _localNetGateway.SubNetMask + "_0_" +
NetGateway.GetConnectorDataKey(_localNetGateway.RealIpComputer);

      //Se arma el mensaje
            MessageData responseMessage =
MessageTemplates.GrantedConnectionRequest(messageData);
            responseMessage.MessageContent = response;
            //Se envía el mensaje a la red
            return CommunicationMessage.SendMessage(responseMessage);
        }
```

- Método que agrega una conexión externa, recibe como parámetros los objetos de conexión local y los objetos de conexión remota

```
        public void AddNetConnection(ConnectorData localConnector,
ConnectorData remoteGateway)
        {
            remoteGateway.GatewayType = GatewayType.Net;
            localConnector.GatewayType = GatewayType.Net;
            localConnector.Port = remoteGateway.Port;
            string key = NetGateway.GetConnectorDataKey(localConnector);
            _routerTable.Add(key, remoteGateway); //Se agrega a la table
de conexiones externas
            _remoteConnections.Add(key, localConnector); //Se agrega a la
table de objetos externos
        }
```

- Método que agrega una conexión externa, recibe como parámetro la configuración de red

```
        public void AddNetConnection(NetGateway netGateway)
        {
            string key;
      //Obtiene la data de conexión
            ConnectorData connectorData =

NetGateway.GetRemoteConnector(netGateway.RemoteConnections, out key);
            ConnectorData remote = netGateway.LocalConnection;
            key = NetGateway.GetConnectorDataKey(remote.Ip);
            remote.GatewayType = GatewayType.Net;
            connectorData.GatewayType = GatewayType.Net;
            connectorData.Port = remote.Port;
            connectorData.Tag = netGateway;
            remote.Tag = netGateway;
            _routerTable.Add(key, remote); //Se agrega a la talba de ruteo
            _remoteConnections.Add(key, connectorData); //Se agrega a la
table de conexiones remotas
```

```
            _tmpCost.Add(_localKey + key, netGateway.Weight); //Se agrega
ala table temporal de costos

      //Finalmente se agrega a la lista de nodos
            AddPath(_localKey + "/" + _localNetGateway.SubNetMask, key +
"/" + netGateway.SubNetMask, netGateway.Weight);

        }
```

- Método que agrega un nodo a la lista de nodos de ruteo

```
        private void AddPath(string source, string destination, int
weight)
        {
            Path<string> newPath = new Path<string>
                                       {
                                           Cost = weight,
                                           Destination = destination,
                                           Source = source
                                       };
      //Si el nodo no ha sido agregado antes, se lo agrega
            if (!_routerTablePath.Contains(newPath))
                _routerTablePath.Add(newPath);
        }
```

- Método que busca el siguiente nodo al que debe irse, recibe como parámetro el nodo destino

```
        private string GetNextPath(string nextStep)
        {
            const string methodName = "GetNextPath";

            _logger.LogStep(ClassName, methodName, "Buscando camino para
llegar de " + LocalPathKey + " a " + nextStep);

      //Se llama al core de Dijkstra para obtener el siguiente paso.
            LinkedList<Path<string>> results =
DijkstraEngine.CalculateShortestPathBetween(LocalPathKey, nextStep,

_routerTablePath);


            int n = results.Sum(r => r.Cost); //Se obtiene el costo de ir
al nuevo paso
            _logger.LogStep(ClassName, methodName, "Costo transaccional: "
+ n);
            _logger.LogStep(ClassName, methodName, "Número de saltos: " +
results.Count);
            _logger.LogStep(ClassName, methodName, "Siguiente destino: " +
results.First.Value.Destination);
```

```
            return results.First.Value.Destination; //Devuelve el destino
siguiente
        }
```

- Realiza un testeo de las conexiones remotas

```
        public void TestConnections()
        {
            const string methodName = "TestConnections";
            List<string> remotesToDelete = new List<string>(); //Lista de
nodos a eliminar por pérdida de conexión
            lock (_remoteConnections) //Se bloque la lista de conexiones
remotas para que no exista inconsistencia de información
            {
      //Para cada element de la lista de conexiones remotas
                foreach (KeyValuePair<string, ConnectorData>
remoteConnection in _remoteConnections)
                {
                    if (remoteConnection.Value.Port > 0) //Si el Puerto es
diferente de cero
                    {
                        ConnectorData gateway;
                        MessageData requestMessage;
//Se obtiene la configuración de destino
                        if (remoteConnection.Value.GatewayType ==
GatewayType.Host)
                        {
                            gateway =
_remoteConnections[remoteConnection.Key];
                        }
                        else
                        {
                            gateway = _routerTable[remoteConnection.Key];

                        }
             //Se crea el mensaje a enviar
                        requestMessage =
MessageTemplates.GetConnectionRequest(gateway.Ip, gateway.Port,

_localIpAddress, _localPort,

gateway.RemoteIp,

_localNetGateway.RealIpComputer,

_localLongPathKey);

                        if (!SendRouteTable(requestMessage)) //Si el envoi
es incorrecto
                        {
             //Se agrega a la lista de nodos a eliminar
                            remotesToDelete.Add(remoteConnection.Key);
                        }
                        else
                            remoteConnection.Value.Client = new
TcpClient();
```

```
                    }
                }

//Finalmente se eliminan de las conexiones externas los nodos a eliminar
                if (remotesToDelete.Count > 0)
                {
                    foreach (string sin remotesToDelete)
                    {
                        RemoveKeyConnection(s);
                        _logger.LogStep(ClassName, methodName,
                                        "Se cierra la conexión con el
punto " + s +
                                        " porque no se ha podido
establecer una conexión.");

                    }
                }
            }
        }
```

- Método que remueve un nodo externo

```
        private void RemoveKeyConnection(string s)
        {
            _remoteConnections[s].Client = null;
            if (_remoteConnections[s].GatewayType == GatewayType.Host)
            {
                _remoteConnections[s].Port = 0;
            }
            else if (_remoteConnections[s].GatewayType == GatewayType.Net)
            {
                lock (_routerTablePath) //Se bloque el diccionario de table de
ruteo
                {
              //Se crea una lista de nodosa a eliminar
                    List<Path<string>> pathsToDelete = new List<Path<string>>();
       //Para cada element de los nodos habilitados
                    foreach (Path<string> path in _routerTablePath)
                    {
                        string id = s + "/" +
((NetGateway)_remoteConnections[s].Tag).SubNetMask;
                        if (path.Destination.Equals(id) ||
                            path.Source.Equals(id))
                        { // Si la clave coincide con el parámetro recibido
entonces se agrega a la lista de nodos a eliminar
                            pathsToDelete.Add(path);
                        }
                    }
       //Finalmente se eliminan los nodos
                    foreach (Path<string> path in pathsToDelete)
                    {
                        _routerTablePath.Remove(path);
                    }
                }

       //Y se elimina de todo diccionario dentro del sistema
                _remoteConnections.Remove(s);
                _routerTable.Remove(s);
                _tmpCost.Remove(_localKey + s);
            }
        }
```

Anexo 3: Clase CommunicationMessage.cs

Clase que maneja las comunicaciones

- Variables globales

```
        //Timeout de lectura de datos
        private const int ReadTimeout = 10000;
        //Timeout de escritura de datos
        private const int WriteTimeout = 10000;
```

- Método que establece la comunicación y envía el mensaje a la red

```
        public static bool SendMessage(MessageData messageData)
        {
            Logger logger = new Logger(LogStyle.InFormal, "RouterClient");
            const string methodName = "SendMessage";
            try
            {
                //bool remote = messageData.MessageFrom.Equals("R");
                //Crea una nueva instancia de comunicación TCP
                TcpClient tcpClient = new TcpClient();
                //Se conecta con el equipo receptor

tcpClient.Connect(MessageTemplates.ConvertToIpAddress(messageData.RealDestinat
ionIp), messageData.RemotePort);

                //Extrae el stream de red
                NetworkStream serverStream = tcpClient.GetStream();
                serverStream.ReadTimeout = ReadTimeout;
                serverStream.WriteTimeout = WriteTimeout;

                //byte[] outStream =
System.Text.Encoding.ASCII.GetBytes(messageData.GetString() + "$");
                //Transforma el objeto a envia en arreglo de bytes
                byte[] outStream =
System.Text.Encoding.ASCII.GetBytes(messageData.GetString());
                //Escribe en el stream de red
                serverStream.Write(outStream, 0, outStream.Length);
                //Libera el stream
                serverStream.Flush();

                logger.LogStep(ClassName, methodName,
                               "1 Se ha enviado satisfactoriamente el mensaje
a " +
                               messageData.RemoteIpAddress + ": " +
messageData.GetString());
                if (!"F".Equals(messageData.MessageType))
                    logger.LogStep(ClassName, methodName,
                                   "Contenido: " +

messageData.DecodedMessage(messageData.MessageContent));
```

```
         //Cierra la comunicación y retorna verdadero
                tcpClient.Close();
                return true;
            }
            catch (Exception exception)
            {
                //En caso de existir algún error escribe la excepción en la
consola y retorna falso
                logger.LogError(ClassName, exception);
                return false;
            }
        }
```

Anexo 4: Clase EncodeManager.cs

Encriptación

- Convierto la cadena y la clave en arreglos de bytes para poder usarlas en las funciones de encriptacion

```
            byte[] cadenaBytes = Encoding.UTF8.GetBytes(cadena);
            byte[] claveBytes = Encoding.UTF8.GetBytes(clave);
```

- Creo un objeto de la clase Rijndael

```
            RijndaelManaged rij = new RijndaelManaged();
```

- Configuro para que utilice el modo ECB

```
            rij.Mode = CipherMode.ECB;
```

- Configuro para que use encriptación de 256 bits.

```
            rij.BlockSize = 256;
```

Declaro que si necesitara más bytes agregue ceros.

```
            rij.Padding = PaddingMode.Zeros;
```

- Declaro un encriptador que use mi clave secreta y un vector de inicializacion aleatorio

```
            ICryptoTransform encriptador;
            encriptador = rij.CreateEncryptor(claveBytes, rij.IV);
```

- Declaro un stream de memoria para que guarde los datos encriptados a medida que se van calculando

```
            MemoryStream memStream = new MemoryStream();
```

- Declaro un stream de cifrado para que pueda escribir aquí la cadena a encriptar. Esta clase utiliza el encriptador y el stream de memoria para realizar la encriptación y para almacenarla

```
            CryptoStream cifradoStream;

            cifradoStream = new CryptoStream(memStream, encriptador,
CryptoStreamMode.Write);
```

- Escribo los bytes a encriptar. A medida que se va escribiendo se va encriptando la cadena

```
            cifradoStream.Write(cadenaBytes, 0, cadenaBytes.Length);
```

- Aviso que la encriptación se terminó

```
            cifradoStream.FlushFinalBlock();
```

- Convertimos la data encriptada en un arreglo de bytes.

```
            byte[] cipherTextBytes = memStream.ToArray();
```

- Cierro los dos streams creados

```
            memStream.Close();
            cifradoStream.Close();
```

- Convierto el resultado en base 64 para que sea legible y devuelvo el resultado

```
            return Convert.ToBase64String(cipherTextBytes);
```

Desencriptación

- Convierto la cadena y la clave en arreglos de bytes para poder usarlas en las funciones de encriptación. En este caso la cadena la convierta usando base 64 que es la codificación usada en el método encriptar

```
            byte[] cadenaBytes = Convert.FromBase64String(cadena);
            byte[] claveBytes = Encoding.UTF8.GetBytes(clave);
```

- Creo un objeto de la clase Rijndael, Configuro para que utilice el modo ECB. Configuro para que use encriptación de 256 bits y Declaro que si necesitara mas bytes agregue ceros.

```
            RijndaelManaged rij = new RijndaelManaged();
            rij.Mode = CipherMode.ECB;
            rij.BlockSize = 256;
            rij.Padding = PaddingMode.Zeros;
```

- Declaro un desencriptador que use mi clave secreta y un vector de inicialización aleatorio

```
            ICryptoTransform desencriptador;
            desencriptador = rij.CreateDecryptor(claveBytes, rij.IV);
```

- Declaro un stream de memoria para que guarde los datos encriptados

```
            MemoryStream memStream = new MemoryStream(cadenaBytes);
```

- Declaro un stream de cifrado para que pueda leer de aquí la cadena a desencriptar. Esta clase utiliza el desencriptador y el stream de memoria para realizar la desencriptación

```
            CryptoStream cifradoStream;
            cifradoStream = new CryptoStream(memStream, desencriptador,
CryptoStreamMode.Read);
```

- Declaro un lector para que lea desde el stream de cifrado. A medida que vaya leyendo se ira desencriptando.

```
            StreamReader lectorStream = new StreamReader(cifradoStream);
```

- Leo todos los bytes y lo almaceno en una cadena

```
            string resultado = lectorStream.ReadToEnd();
```

- Cierro los dos streams creados

```
            memStream.Close();
            cifradoStream.Close();
```

- Devuelvo la cadena

```
            return resultado;
```

Anexo 5: Clase MessageData.cs

- Métdo que parsea el mensaje

```
        private void ParseMessage()
        {
            const string methodName = "ParseMessage";

            _localIpAddress = _originalMessage.Substring(0, 12); //Obtiene
la ip local
            _localPort = Convert.ToInt32(_originalMessage.Substring(12,
6)); //Obtiene el Puerto local

            _remoteIpAddress = _originalMessage.Substring(18, 12);
            _remotePort = Convert.ToInt32(_originalMessage.Substring(30,
6));

            if (Convert.ToInt32(_originalMessage.Substring(36, 1)) == 1)
```

```
            {
                _fullData = false;

                _currentSecuencialData =
Convert.ToInt32(_originalMessage.Substring(37, 8));
                _nextSecuencialData =
Convert.ToInt32(_originalMessage.Substring(45, 8));

                _responseMethod = _originalMessage.Substring(53, 3);
                _messageType = _originalMessage.Substring(56, 1);

                _messageFrom = _originalMessage.Substring(57, 1);

                _netOwner = _originalMessage.Substring(58, 15);

                RealSourceIp = _originalMessage.Substring(73, 12);

                _realDestinationIp = _originalMessage.Substring(85, 12);

                _contentLenght =
Convert.ToInt32(_originalMessage.Substring(97, 8));

                _messageContent =
DecodedMessage(_originalMessage.Substring(105, _contentLenght));
            }
            else
            {
                _fullData = true;
                _responseMethod = _originalMessage.Substring(37, 3);
                _messageType = _originalMessage.Substring(40, 1);

                _messageFrom = _originalMessage.Substring(41, 1);

                _netOwner = _originalMessage.Substring(42, 15);

                RealSourceIp = _originalMessage.Substring(57, 12);

                _realDestinationIp = _originalMessage.Substring(69, 12);

                _contentLenght =
Convert.ToInt32(_originalMessage.Substring(81, 8));

                _messageContent =
DecodedMessage(_originalMessage.Substring(89, _contentLenght));
            }
        }
```

- Método que decodifica el mensaje

```
        public string DecodedMessage(string content)
        {
            if ("F".Equals(_messageType))
                return content;

            const string methodName = "DecodedMessage";
            _logger.LogStep(ClassName, methodName, "Decodificando el
contenido del mensaje: " + content);
            //string decodedContent =
EncodeManager.Decode(content.Replace("\0", ""), Constants.VirtueKey);
```

```
            string decodedContent = EncodeManager.Decode(content,
Constants.VirtueKey);
            //decodedContent = decodedContent.Replace("\0", "");
            _logger.LogStep(ClassName, methodName, "Contenido
decodificado: " + decodedContent);
            return decodedContent;
        }
```

- Método que transforma a cadena de caracteres el mensaje

```
 public string GetString()
        {
            string output = _localIpAddress +
Convert.ToString(_localPort).PadLeft(6, '0') + _remoteIpAddress
Convert.ToString(_remotePort).PadLeft(6, '0');
            if (!_fullData)
            {
                output += "1" +
Convert.ToString(_currentSecuencialData).PadLeft(8, '0') +
Convert.ToString(_nextSecuencialData).PadLeft(8, '0') + _responseMethod +
_messageType +
                          _messageFrom + _netOwner + RealSourceIp +
_realDestinationIp +
                          Convert.ToString(_contentLenght).PadLeft(8, '0')
+ _messageContent;
            }
            else
            {
                output += "0" + _responseMethod + _messageType +
_messageFrom + _netOwner + RealSourceIp +
                          _realDestinationIp +
Convert.ToString(_contentLenght).PadLeft(8, '0') + _messageContent;
            }
            return output;
        }
```

Anexo 6: Clase FileSplitter.cs

- Método que divide el archivo en partes

```
   public static Dictionary<int, string> SplitFile(string file)
        {
  Dictionary<int, string> fileData = new Dictionary<int, string>();
            byte[] fileC = File.ReadAllBytes(file);
            string test = Encoding.Default.GetString(fileC);
            int sequential = 1;
            fileData.Add(0, file);
            for (int i = 0; i < test.Length; )
            {
                string substr = "";
                sequential++;
                if (i + MaxSizeOfPartFile <= test.Length)
                    substr = test.Substring(i, MaxSizeOfPartFile);
                else
                    substr = test.Substring(i);
                fileData.Add(sequential, substr);
```

```
                i = i + MaxSizeOfPartFile;
            }
            fileData.Add(1, (sequential + "").PadLeft(8, '0'));
            return fileData;
            }
```

- Método que une las partes de un archivo

```
  public static void WriteFile(Dictionary<int, string> fileData)
        {
           string fileName = Path.GetFileNameWithoutExtension(fileData[0])
+ "Rearmada" + Path.GetExtension(fileData[0]);
            int parts = Convert.ToInt32(fileData[1]);
            string data = "";
            for (int i = 2; i <= parts; i++)
            {
                data += fileData[i];
            }
            byte[] reverseC = Encoding.Default.GetBytes(data);
            File.WriteAllBytes(fileName, reverseC);
        }
```